High available GNU/Linux systems

Agustín Rivero de la Cruz
ETIS

Ignasi Rius Ferrer

January 8, 2014

Dedication

This work is dedicated to my wife, Eva, who has shown a lot of patience while I was doing this work, and who has take care of our sons. And also to them, Isaac, which is only three and didn't understand why I wasn't playing with him while I was at home, and Dario, who was born just a month ago.

I am very grateful to all of them.

Summary

This paper covers High availability methods available in GNU/Linux environments. It is oriented towards a practical point of view where companies try to have very short downtimes and reduce data loss in their services.

It will be written about everything needed to reach those goals. First, it will cover data allocation of servers, both its service data and its own Operating System. Then it will inform about network connectivity of those servers at both layer two and layer three levels. Next, it will expose some ways to have available Operating Systems, normally using virtual machines. Finally, it will cover some methods that can be used directly in applications to be able to keep services alive, even if a server fails or becomes unreacheable.

This paper is divided into chapters, each one will have three sections: a general overview of considerations to be taken for every chapter scope, a detailed section of how can be GNU/Linux used to achive those goals and, in the end, an explanatory example using GNU/Linux systems.

There is a final chapter with a summary and some conclusions.

Contents

List of Figures

Chapter 1

General Overview on High Availability

1.1 Overview

Nowadays IT services heavily relay on availability of networked applications[25, 31]. In turn those applications depend on hardware that suffers from outages due to human interaction or instability and expiration of computer equipments. Sometimes hardware crashes, and some other times someone hangs a system voluntarily or involuntarily.

To avoid this service outages, organizations can relay on better hardware: dual powered, rugged, rock-solid components, RAID disks, good levels of temperature and humidity, etc. Those organizations can also benefit from architectures and applications designed to avoid service downtime and data loss.

It is needed to protect services from individual computer component crashes, from attacks coming from Internet and from poor handling of system owners. There are maintenance works that can compromise service availability and, if an outage happens, it is possible that valuable data is lost, so it is important to be able to recover as much data as possible if this happens.

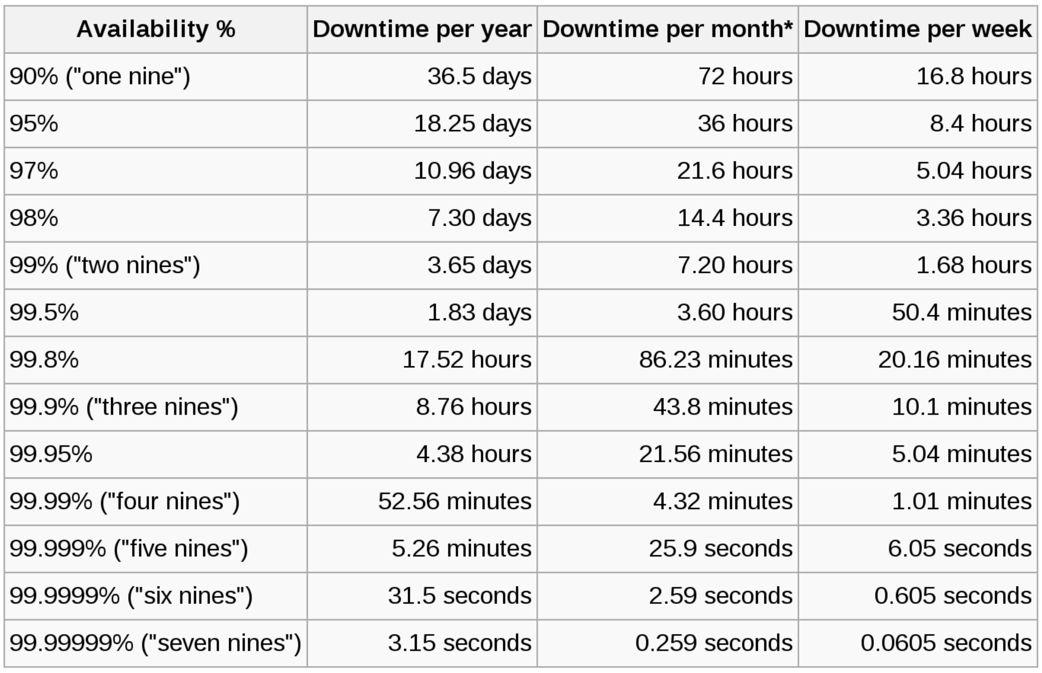
This whole document explains the goals sought by named High Availability techniques, those used to provide service continuity at the IT world. Initially it will talk about general considerations, but all examples will cover the achievement of this objectives from a GNU/Linux point of view.

1.2 Goals

The main goal[6] is to lower costs due to service unavailability and data loss. The cost of downtime impacts on productivity, reputation revenue and margin of organization business.

There are many ways to measure downtime, one of them is a percentage of service availability per time unit. Normally, this time unit is a year, and the higher the percentage, the lower the downtime. Figure 1.1[57] shows a table with the equivalence between percentage and downtime per unit of time.

It is easy to notice that this measurement is not enough to have an idea of service quality. This *percentage availability* is usually agreed in business contracts that like to offer a good number of *nines*. It is clearly not the same to have near to 9 hours of downtime once a year, than one ten-minute outage per month.



Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
95%	18.25 days	36 hours	8.4 hours
97%	10.96 days	21.6 hours	5.04 hours
98%	7.30 days	14.4 hours	3.36 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% ("seven nines")	3.15 seconds	0.259 seconds	0.0605 seconds

Figure 1.1: Availability percentage

Some businesses won't tolerate twelve outages per year and some others won't accept a 9-hour outage, so this is a poor way to measure unavailability. Thus, some other concepts appear when it is needed to define service availability, a few stated by vendors and service providers are:

- Mean Time between Failures (MTBF)[48]. Is the average time between a failure and next one, i.e. the expected average time to wait until next failure occurs.
- Mean Time To Recover (MTTR)[49]. Opposite to MTBF, this is the average time for outages, i.e. the expected average time to wait between service failure and restoration.
- Data loss limits[6, 31]. Possibility and data loss quantification in outages that normally is measured in time units. Organizations can offer a data loss limit of a maximum period of time.

All these measurements can be some of the ones offered in contracts to warrant a service. It is the so called Service Level Agreement[57].

1.3 Procedure to reach those goals

Every organization that wants to narrow service unavailability needs to follow this procedure[6, 31]:

- Complete a business impact analysis and identify and categorize critical business processes with HA requirements. All business processes should be identified. Also, there should be a quantification of the risk for outages of these processes. These evaluations categorize the processes depending on impact severity of outages, and services provided by an organization should be partitioned into desired levels of availability. For example, a web service needs the website to be all the time available, but product delivery can be delayed for some time. An example of this aggregation of

Availability Levels is summarized in figure 1.2. This figure [6] shows Availability Levels understood by Citrix to offer their clustering services. This company is following four levels of availability ranged from more (AL-4) to less (AL-0) critical, defined by IDC terms in 2009. In following lines RTO and RPO will be explained.

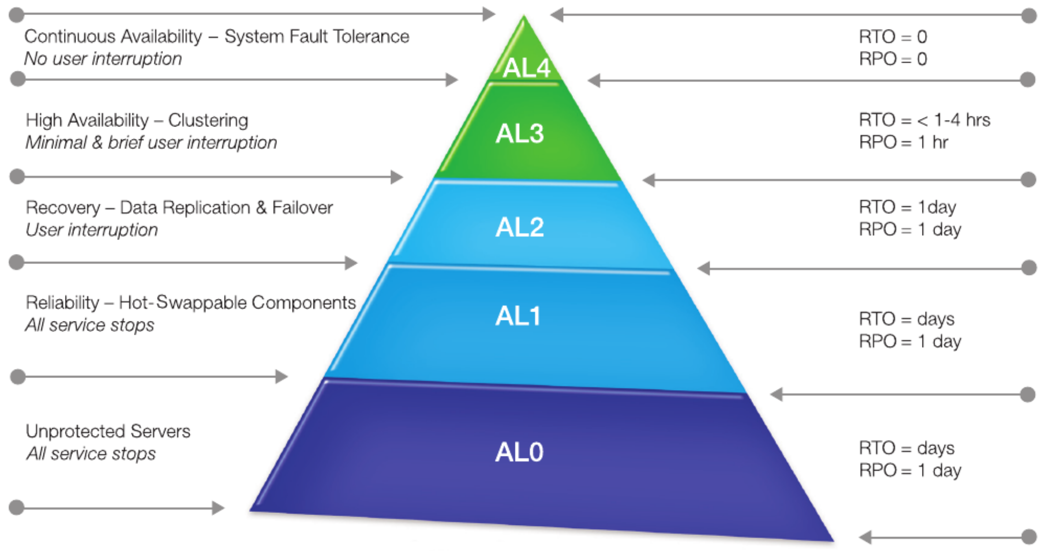


Figure 1.2: Availability Levels

- Formulate cost of downtime. Organizations must know the costs of an outage on every particular business category identified before. The costs can be produced because business halting lowers revenue, partner relationship can be affected, reputation can be lowered and there can be legal consequences.
- Establish RTO and RPO goals. As costs had been evaluated, a Recovery Time Objective (RTO) and a Recovery Point Objective (RPO) should be established. RTO is related to maximum MTTR[49] accepted and RPO to maximum data loss permitted. This levels of RTO and RPO heavily depend on the Availability Level, as defined in previous figure 1.2.
- Understand goals for manageability, TCO and ROI. Once objectives are established, the organization may need to improve workers knowledge, acquire equipments or rent new services. It should be taken in cosideration:
  - Manageability, which is the real capability of the company to achieve these objectives, may be some improvements are necessary to achieve them.
  - Total Cost of Ownership (TCO) and Return On Investment (ROI). Are needed equipments and services worth paying now that costs of downtime are known? They should.

Chapter 2

Storage

2.1 Ways to provide high available storage

This section it will be addressed on service availability uptime. When it is thought about high available storage, it is also thought about a file system that is always ready for an Operating System to be used.

Traditionally computers had disk drives but disk drives may fail. Fortunatelly some techniques exists to overcome those limitations.

2.1.1 Local RAID

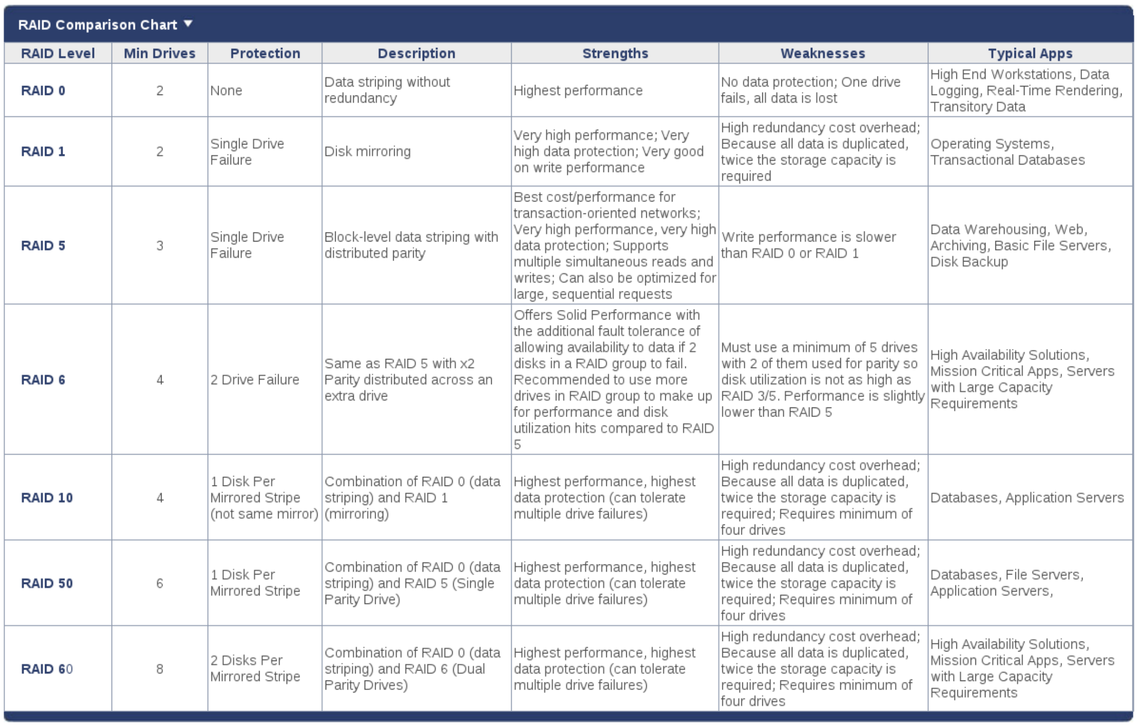
Redundant Array of Inexpensive Disks provide a mechanism to be able to suffer a disk failure without losing file system and data availability. The system manages an array of disks and some disks contain all the information needed to know the content of a possible missing disk. Normally a bitwise sum of all disks is saved on a redundancy disk, so in case of any disk failure this disk can be used to calculate the data of the missing one. This is the definition or RAID level 5. More levels exist in order to provide better redundancy or performance. Figure 2.1 shows a list of RAID levels, a description and minimum number of drives to build it[13].

But RAID disks only allow data to be accessible from a single equipment, if this computer fails, no other can reach the data to provide high availability of the content, so next sections will show how to offer high available storage systems.

2.1.2 Network block device solutions

A block device can be offered to networked systems. This device is seen by a computer like a local drive, which can be partitioned, formatted and used. All storage vendors offer turn-key solutions using this technique[27, 34]. It's a multi-purpose general solution.

Those drives can be used even to boot the Operating System by a disk-less computer. For this to work, it is also needed a BIOS that understands network booting, usually PXE (DHCP/TFTP) or Host Bus Adapters (HBA) are used to present remote drives as local ones.



RAID Comparison Chart ▾

RAID Level	Min Drives	Protection	Description	Strengths	Weaknesses	Typical Apps
RAID 0	2	None	Data striping without redundancy	Highest performance	No data protection; One drive fails, all data is lost	High End Workstations, Data Logging, Real-Time Rendering, Transitory Data
RAID 1	2	Single Drive Failure	Disk mirroring	Very high performance; Very high data protection; Very good on write performance	High redundancy cost overhead; Because all data is duplicated, twice the storage capacity is required	Operating Systems, Transactional Databases
RAID 5	3	Single Drive Failure	Block-level data striping with distributed parity	Best cost/performance for transaction-oriented networks; Very high performance, very high data protection; Supports multiple simultaneous reads and writes; Can also be optimized for large, sequential requests	Write performance is slower than RAID 0 or RAID 1	Data Warehousing, Web, Archiving, Basic File Servers, Disk Backup
RAID 6	4	2 Drive Failure	Same as RAID 5 with x2 Parity distributed across an extra drive	Offers Solid Performance with the additional fault tolerance of allowing availability to data if 2 disks in a RAID group to fail. Recommended to use more drives in RAID group to make up for performance and disk utilization hits compared to RAID 5	Must use a minimum of 5 drives with 2 of them used for parity so disk utilization is not as high as RAID 3/5. Performance is slightly lower than RAID 5	High Availability Solutions, Mission Critical Apps, Servers with Large Capacity Requirements
RAID 10	4	1 Disk Per Mirrored Stripe (not same mirror)	Combination of RAID 0 (data striping) and RAID 1 (mirroring)	Highest performance, highest data protection (can tolerate multiple drive failures)	High redundancy cost overhead; Because all data is duplicated, twice the storage capacity is required; Requires minimum of four drives	Databases, Application Servers
RAID 50	6	1 Disk Per Mirrored Stripe	Combination of RAID 0 (data striping) and RAID 5 (Single Parity Drive)	Highest performance, highest data protection (can tolerate multiple drive failures)	High redundancy cost overhead; Because all data is duplicated, twice the storage capacity is required; Requires minimum of four drives	Databases, File Servers, Application Servers,
RAID 60	8	2 Disks Per Mirrored Stripe	Combination of RAID 0 (data striping) and RAID 6 (Dual Parity Drives)	Highest performance, highest data protection (can tolerate multiple drive failures)	High redundancy cost overhead; Because all data is duplicated, twice the storage capacity is required; Requires minimum of four drives	High Availability Solutions, Mission Critical Apps, Servers with Large Capacity Requirements

Figure 2.1: Raid Levels

The most used protocol for block device presentation is ISCSI, again all vendors offer this protocol. There are some alternatives, for example GNU/Linux can use ATA over Ethernet (AoE), but ISCSI is widely extended and offers the best performance.

2.2 Provide Data Recovery

This is the only chapter addressing Disaster Recovery and data loss prevention. In order to provide Data Recovery it is needed to have copies of data, so, in case of data corruption or loss, it is possible to recover it at least from a point in time not much ago.

There should be copies of important data outside from the active production systems. There are many levels where this can be achieved. At block device level there can be a network copy, at file level there can also exist copies and at application level, for example a database, there can be copies of data.

Generally speaking, there are two main kinds of data backup strategies: versioned backups or fast recovery ones.

2.2.1 Versioned backups

Organizations can create backups of data in any medium (disk drives, optical drives, network copies...) at a point in time. It's usual to make a copy in night hours when services can be stopped or performance can be reduced because there are less customers accessing them.

This can be enough for Application Levels with low RPO aim. If copies are made daily, then recover point can be a day before. For small losses of data a fast recovery procedure should be used.

With this technique if there is a failure in the data source, normally a human interaction is needed to recover the service at last backup point in time.

2.2.2 Fast recovery

For smallest data loss, active/passive procedures can be used. There can be an passive or standby system receiving all data from active one. When active systems fails, there should be a mechanism that redirects all customer requests to passive (from now on active) system. The loss of data will be the one produced by active system but not yet received by passive one. For a database, for example, this can be in the order of milliseconds of produced data, and some databases can work synchronously, i.e. applications working in active system don't get the confirmation from their transactions until passive database receives and processes it.

Databases aren't the only service that can benefit from this techniques, but they work at application level, so customers can notice no failure at all. A website for example can detect a database interruption and autonomously change connection to passive servers.

Fast recovery can also be used at block device level. A block device can be network-copied in real-time and file services can be restored from those standby systems. Previous example (a website) will normally suffer an interruption in case a failure.

2.2.3 Versioned backup versus fast recovery

For low RPO, both techniques should be used. Generally Fast recovery will give a better service (at a higher cost) because recovery time and data loss are very low. But sometimes it's not enough. An advantage of versioned backups is that some versions can be stored and if a mistake is made that deletes some data, standby systems can also delete data. So the best solution is an hybrid system using both techniques: standby systems receiving data at real-time and versioned backups taken from those passive systems that normally can suffer from low performance while making backups. Also application level procedures should be used whenever possible to avoid service interruption (RTO equal to zero)

2.3 The GNU/Linux way

2.3.1 Local RAID

For local RAIDs normally hardware RAID is used because it should provide better performance and lower CPU usage, also good hardware controllers have batteries that power up buffers, so in case of power failure there is enough power to write buffers to disk. But some hardware lacks of this kind of hardware and some inexpensive RAID hardware controllers really are software ones with very poor performance. In this cases a software RAID is recommended. Furthermore hardware controllers are proprietary and disk meta-data is not interchangeable between different hardware controllers, with software RAID this limitation simply doesn't exist as meta-data is controlled by software.

GNU/Linux comes out-of-the-box with good RAID tools, mdadm supports any RAID level from 0 to 6 and any layered combination (1+0, etc)

2.3.2 Network Block Devices

Most used Network Block Device (NBD) is by far ISCSI, it's an standard in the industry and all modern Operating Systems support it. GNU/Linux uses tgt[39] software to provide this protocol support.

Another good NBD is AoE[1] which is fast, reliable and easy to configure, but not as deployed as ISCSI and only supported in GNU/Linux, BSD and Solaris systems.

2.3.3 Network File-systems

Another way to provide file-systems at a higher level are network file-systems. NFS[51], WebDAV[61] or CIFS[52] are the most used ones. This technique doesn't provide a full block device that can be formatted (or even used as swap space), this protocols directly provide files and directories. It's a good way to provide the same files to a lot of hosts simultaneously without needing a shared file-system known at block device level, like GFS2[37] or OCFS2[33] that must be configured between clients. For example for DAV even a web browser is enough to access the files.

For NFS, GNU/Linux uses both kernel and user-space native tools, for WebDAV a web server like Apache httpd can be used, and for CIFS normally Samba is used.

2.3.4 Booting

With no disk, booting is a bit tricky. Hardware Motherboards usually support Ethernet booting by using PXE[54] protocol, in GNU/Linux the process consists in:

1. DHCP protocol provides an IP address and a TFTP server for booting
2. Host downloads from TFTP server a file with a filename of its Ethernet MAC-Address
3. TFTP service provides a system to boot both kernel and initramdisk
4. Host boots kernel and loads initramdisk at initial Root File-system
5. initramdisk continues the boot. It receives as parameters the final root device which can be an ISCSI target, a NFS export or whatever network device commented before

This final point is may be the most difficult to build, a good tool to do it is Dracut[22] which supports ISCSI root, NFS root and more network redundancy support that we will see later.

2.4 Practical Example

2.4.1 Description

It will be built an high available storage block device. Three hosts will be needed, two of them will provide the file system, in active/passive mode, the third host will have the block device formatted and mounted. If active server fails, passive one will become active giving service to client host.

File system servers will be `iscsi-server0` and `iscsi-server1` and client will be `iscsi-client`. The three of them will have Debian Wheezy Operating System and will be KVM virtual machines. The servers will have a partition (`/dev/vda2`) replicated between them using DRBD[10] software. This application will replicate partition `/dev/drbd0` that will be a physical device for a ISCSI network drive. `iscsi-server0` will be the primary server and heartbeat[23] software will teach `iscsi-server1` to give service in case of `iscsi-server0` failure. `iscsi-client` will have this network drive formatted and mounted on `/mnt`.

When it is mounted, active ISCSI server will be stopped and it will be seen a fast recovery, in figure 2.2 fast recovery and redundancy recovery idea can be seen.

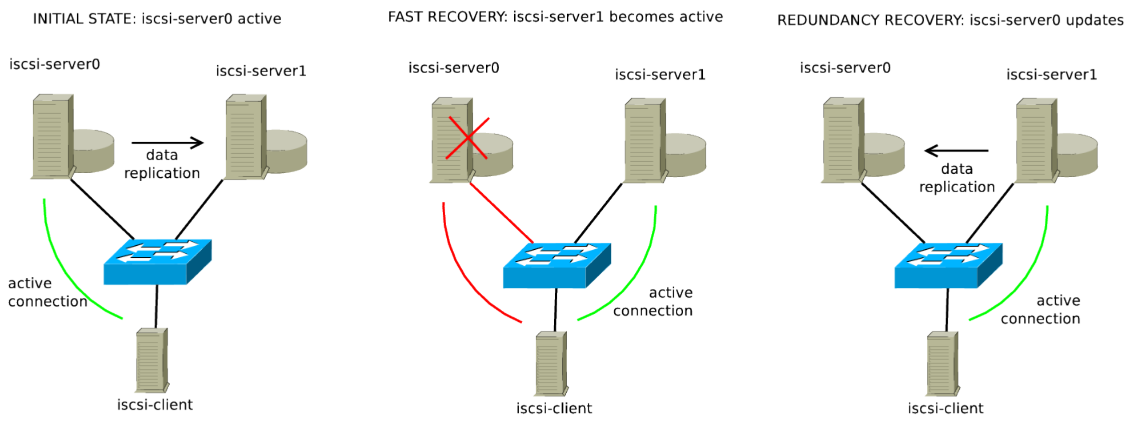


Figure 2.2: ISCSI example recoveries

2.4.2 Configuration

Virtual machines will be installed but without any software needed.

As all hosts need to connect each other, a DNS would be desirable, but instead, an identical `/etc/hosts` file will be configured on all hosts. This will be its content:

```
127.0.0.1       localhost

192.168.103.99  iscsi-server0
192.168.103.91  iscsi-server1
192.168.103.36  iscsi-server-ha
192.168.103.93  iscsi-client
```

2.4.2.1 DRBD

First, it will needed to install DRBD software to have both servers with a synchronized partition. There will be a primary server and a secondary one. Only primary servers can write on a device, secondaries only write to disk what is received from primaries.

```
# apt-get install drbd8-utils
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  heirloom-mailx
```

```
Suggested packages:
  heartbeat exim4 mail-transport-agent
The following NEW packages will be installed:
  drbd8-utils heirloom-mailx
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 524 kB of archives.
After this operation, 1,141 kB of additional disk space will be used.
Do you want to continue [Y/n]?
Get:1 http://ftp.fi.debian.org/debian/ wheezy/main heirloom-mailx amd64 12.5-2 [274 kB]
Get:2 http://ftp.fi.debian.org/debian/ wheezy/main drbd8-utils amd64 2:8.3.13-2 [250 kB]
Fetched 524 kB in 1s (269 kB/s)
Selecting previously unselected package heirloom-mailx.
(Reading database ... 18431 files and directories currently installed.)
Unpacking heirloom-mailx (from .../heirloom-mailx_12.5-2_amd64.deb) ...
Selecting previously unselected package drbd8-utils.
Unpacking drbd8-utils (from .../drbd8-utils_2%3a8.3.13-2_amd64.deb) ...
Processing triggers for man-db ...
Setting up heirloom-mailx (12.5-2) ...
update-alternatives: using /usr/bin/heirloom-mailx to provide /usr/bin/mailx (mailx) in auto mode
Setting up drbd8-utils (2:8.3.13-2) ...
```

Now two files should be eddited in those servers. One is `/etc/drbd.d/global_common.conf` that will have this content:

```
global {
        usage-count no;
}

common {
        protocol C;
}
```

There are three types of protocols[11]:

1. protocol A: Less safe but best performance. Write IO reported as completed when reached local disk and TCP send buffer.
2. protocol B: Safer. Write IO reported as completed when reached local disk and remote buffer cache.
3. protocol C: Safest and the only synchronous protocol, RPO equal to zero. Write IO reported as completed when reached both local and remote disk.

Then it is needed to add a resource, a file named `/etc/drbd.d/r0.conf` will be created with this content:

```
resource r0 {
  device    /dev/drbd0;
  disk      /dev/vda2;
  meta-disk internal;
  on iscsi-server0 {
    address   192.168.103.99:7789;
  }
  on iscsi-server1 {
    address   192.168.103.91:7789;
  }
}
```

Then it is needed to initialize the partition on both servers:

```
# drbdadm create-md r0
The server’s response is:
```

```
node already registered
Writing meta data...
initializing activity log
NOT initialized bitmap
New drbd meta data block successfully created.
success
```

Now drbd service can be started up. It is needed to be started on both nodes for the startup to finish:

```
# /etc/init.d/drbd start
[ ok ] Starting DRBD resources:[ d(r0) n(r0) ]......
```

It can be seen this at `/proc/drbd` to see DRBD status:

```
# cat /proc/drbd
version: 8.3.11 (api:88/proto:86-96)
srcversion: F937DCB2E5D83C6CCE4A6C9
 0: cs:Connected ro:Secondary/Secondary ds:Inconsistent/Inconsistent C r-----
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:225236
```

It is needed to initialize the partition and label one of the servers as primary, secondary will then synchronize the whole disk (like a mirroring RAID initialization). On desired primary server, it is written:

```
# drbdadm -- --overwrite-data-of-peer primary r0
```

Then synchronization has started as it can be seen on `/proc/drbd`:

```
# cat /proc/drbd
version: 8.3.11 (api:88/proto:86-96)
srcversion: F937DCB2E5D83C6CCE4A6C9
 0: cs:SyncSource ro:Primary/Secondary ds:UpToDate/Inconsistent C r-----
    ns:1872 nr:0 dw:0 dr:2536 al:0 bm:0 lo:0 pe:6 ua:0 ap:0 ep:1 wo:f oos:223388
        [>....................] sync'ed:  1.9% (223388/225236)K
        finish: 0:13:01 speed: 264 (264) K/sec
```

Now `iscsi-server0` is primary server and its data-source is up to date. Contrary, `iscsi-server1` is secondary and, by now, inconsistent.

And if some time passes it will be seen it has completed syncing and `iscsi-server1` is updated:

```
# cat /proc/drbd
version: 8.3.11 (api:88/proto:86-96)
srcversion: F937DCB2E5D83C6CCE4A6C9
 0: cs:Connected ro:Secondary/Primary ds:UpToDate/UpToDate C r-----
    ns:0 nr:225236 dw:225236 dr:0 al:0 bm:14 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

And now that both servers are up to date, they should be turn into primary. It's important to say that both servers cannot write at the same time unless who writes knows there are two paths to do it. A typical file-system like ext3, ext4 or xfs is not suited for two clients mounting the file-system at once. Also both DRBD servers cannot mount this file-system simultaneously because in this way data will become inconsistent. For this to work a shared file-system like GFS[37] or OCFS2[33] is needed. This documentation is beyond the scope of this document, but the bibliography can be followed to find more information.

So for turning them both primary `/etc/drbd.d/global_common.conf` should be changed to:

```
global {
        usage-count no;
}
```

```
common {
        protocol C;
        net {
                allow-two-primaries;
        }
}
```

And `/etc/drbd.d/r0.res` to:

```
resource r0 {
  device    /dev/drbd0;
  disk      /dev/vda2;
  meta-disk internal;
  startup {
    become-primary-on both;
  }
  on iscsi-server0 {
    address   192.168.103.99:7789;
  }
  on iscsi-server1 {
    address   192.168.103.91:7789;
  }
}
```

And service `drbd` should be stopped and started on both servers.

On `iscsi-server0`:

```
root@iscsi-server0:~# /etc/init.d/drbd stop
[ ok ] Stopping all DRBD resources:.
```

On `iscsi-server1`:

```
root@iscsi-server1:~# /etc/init.d/drbd stop
[ ok ] Stopping all DRBD resources:.
```

on `iscsi-server0`:

```
root@iscsi-server0:~# /etc/init.d/drbd start
[....] Starting DRBD resources:[ d(r0) n(r0) ]..........
```

And on `iscsi-server1` while `iscsi-server0` is waiting for the other node:

```
root@iscsi-server1:~# /etc/init.d/drbd start
[ ok ] Starting DRBD resources:[ d(r0) n(r0) ].
```

Now both of them are primary and up to date. It can be seen at `/proc/drbd`:

```
# cat /proc/drbd
version: 8.3.11 (api:88/proto:86-96)
srcversion: F937DCB2E5D83C6CCE4A6C9
 0: cs:Connected ro:Primary/Primary ds:UpToDate/UpToDate C r-----
    ns:0 nr:0 dw:0 dr:664 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0
```

Now `/dev/drbd0` is ready on both nodes. But again, it should be remembered that *normal* file-systems are not suited for two clients simultaneously.

2.4.2.2 ISCSI target

The software needed is on a package named `tgt`. It should be installed on both servers:

```
# apt-get install tgt
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libclass-isa-perl libconfig-general-perl libibverbs1 librdmacm1 libsgutils2-2 libswitch-perl perl
  perl-modules sg3-utils
Suggested packages:
  perl-doc libterm-readline-gnu-perl libterm-readline-perl-perl make libpod-plainer-perl
The following NEW packages will be installed:
  libclass-isa-perl libconfig-general-perl libibverbs1 librdmacm1 libsgutils2-2 libswitch-perl perl
  perl-modules sg3-utils tgt
0 upgraded, 10 newly installed, 0 to remove and 0 not upgraded.
Need to get 8,973 kB of archives.
After this operation, 34.5 MB of additional disk space will be used.
Do you want to continue [Y/n]?
Get:1 http://ftp.fi.debian.org/debian/ wheezy/main libclass-isa-perl all 0.36-3 [12.3 kB]
Get:2 http://ftp.fi.debian.org/debian/ wheezy/main perl-modules all 5.14.2-21+deb7u1 [3,440 kB]
Get:3 http://ftp.fi.debian.org/debian/ wheezy/main perl amd64 5.14.2-21+deb7u1 [4,407 kB]
Get:4 http://ftp.fi.debian.org/debian/ wheezy/main libswitch-perl all 2.16-2 [21.0 kB]
Get:5 http://ftp.fi.debian.org/debian/ wheezy/main libconfig-general-perl all 2.50-1 [70.9 kB]
Get:6 http://ftp.fi.debian.org/debian/ wheezy/main libsgutils2-2 amd64 1.33-1 [80.0 kB]
Get:7 http://ftp.fi.debian.org/debian/ wheezy/main sg3-utils amd64 1.33-1 [651 kB]
Get:8 http://ftp.fi.debian.org/debian/ wheezy/main libibverbs1 amd64 1.1.6-1 [35.3 kB]
Get:9 http://ftp.fi.debian.org/debian/ wheezy/main librdmacm1 amd64 1.0.15-1+deb7u1 [19.0 kB]
Get:10 http://ftp.fi.debian.org/debian/ wheezy/main tgt amd64 1:1.0.17-1 [237 kB]
Fetched 8,973 kB in 22s (392 kB/s)
Selecting previously unselected package libclass-isa-perl.
(Reading database ... 18495 files and directories currently installed.)
Unpacking libclass-isa-perl (from .../libclass-isa-perl_0.36-3_all.deb) ...
Selecting previously unselected package perl-modules.
Unpacking perl-modules (from .../perl-modules_5.14.2-21+deb7u1_all.deb) ...
Selecting previously unselected package perl.
Unpacking perl (from .../perl_5.14.2-21+deb7u1_amd64.deb) ...
Selecting previously unselected package libswitch-perl.
Unpacking libswitch-perl (from .../libswitch-perl_2.16-2_all.deb) ...
Selecting previously unselected package libconfig-general-perl.
Unpacking libconfig-general-perl (from .../libconfig-general-perl_2.50-1_all.deb) ...
Selecting previously unselected package libsgutils2-2.
Unpacking libsgutils2-2 (from .../libsgutils2-2_1.33-1_amd64.deb) ...
Selecting previously unselected package sg3-utils.
Unpacking sg3-utils (from .../sg3-utils_1.33-1_amd64.deb) ...
Selecting previously unselected package libibverbs1.
Unpacking libibverbs1 (from .../libibverbs1_1.1.6-1_amd64.deb) ...
Selecting previously unselected package librdmacm1.
Unpacking librdmacm1 (from .../librdmacm1_1.0.15-1+deb7u1_amd64.deb) ...
Selecting previously unselected package tgt.
Unpacking tgt (from .../tgt_1%3a1.0.17-1_amd64.deb) ...
Processing triggers for man-db ...
Setting up libclass-isa-perl (0.36-3) ...
Setting up libsgutils2-2 (1.33-1) ...
Setting up sg3-utils (1.33-1) ...
Setting up libibverbs1 (1.1.6-1) ...
Setting up librdmacm1 (1.0.15-1+deb7u1) ...
Setting up perl-modules (5.14.2-21+deb7u1) ...
Setting up perl (5.14.2-21+deb7u1) ...
update-alternatives: using /usr/bin/prename to provide /usr/bin/rename (rename) in auto mode
Setting up libconfig-general-perl (2.50-1) ...
Setting up tgt (1:1.0.17-1) ...
Setting up libswitch-perl (2.16-2) ...
```

Debian lacks of an init script and one provided by Debian Bug tracking system[16] can be used. So, now the service can be started:

```
# /etc/init.d/tgt start
[ ok ] Starting target framework daemon: tgtd.
```

Now a new target named `iqn.2013-11.edu.uoc:ha-disk` can be added as first target id (1):

```
# tgtadm --lld iscsi --op new --mode target --tid 1 -T iqn.2013-11.edu.uoc:ha-disk
```

And finally our ha disk can be added to it:

```
# tgtadm --lld iscsi --op new --mode logicalunit --tid 1 --lun 1 -b /dev/drbd0
```

This command can be issued to have a look at ISCSI exported targets:

```
# tgtadm --lld iscsi --op show --mode target
Target 1: iqn.2013-11.edu.uoc:ha-disk
    System information:
        Driver: iscsi
        State: ready
    I_T nexus information:
    LUN information:
        LUN: 0
            Type: controller
            SCSI ID: IET     00010000
            SCSI SN: beaf10
            Size: 0 MB, Block size: 1
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: null
            Backing store path: None
            Backing store flags:
        LUN: 1
            Type: disk
            SCSI ID: IET     00010001
            SCSI SN: beaf11
            Size: 231 MB, Block size: 512
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: rdwr
            Backing store path: /dev/drbd0
            Backing store flags:
    Account information:
    ACL information:
```

Now this configuration should be saved, exported to `iscsi-server1` and the service can be stopped, heartbeat will be responsible of starting it in active node:

```
# tgt-admin --dump > /etc/tgt/targets.conf
# scp /etc/tgt/targets.conf iscsi-server1:/etc/tgt/
root@iscsi-server1's password:
```

2.4.2.3 Heartbeat

Heartbeat[23] will define a primary node for serving ISCSI target service. It will undertake the task of starting `tgt` service and adding a heartbeat IP address to one and only one server. A heartbeat IP address is an IP that will only have the active system. All clients will connect to this IP, so they won't need to know the complexity of the cluster nodes, it will just contact that IP address.

First of all, Heartbeat software should be installed on both servers:

```
# apt-get install heartbeat
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  ca-certificates cluster-glue dbus file gawk libcap2 libcfg4 libcib1 libconfdb4 libcoroipcc4
  libcoroipcs4 libcorosync4 libcpg4 libcrmcluster1 libcrmcommon2 libcurl3 libdbus-1-3 libdbus-glib-1-2
  libesmtp6 libevs4 libffi5 libglib2.0-0 libglib2.0-data libheartbeat2 libldap-2.4-2 liblogsys4
  liblrm2 libltdl7 libmagic1 libnet1 libnspr4 libnspr4-0d libnss3 libnss3-1d libopenhpi2 libopenipmi0
  libpcre3 libpe-rules2 libpe-status3 libpengine3 libperl5.14 libpils2 libpload4 libplumb2libplumbgpl2
  libquorum4 librtmp0 libsam4 libsasl2-2 libsasl2-modules libsensors4 libsigsegv2 libsnmp-base
  libsnmp15 libssh2-1 libstonith1 libstonithd1 libsystemd-login0 libtimedate-perl libtotem-pg4
  libtransitioner1 libvotequorum4 libxml2 libxml2-utils libxslt1.1 mime-support openhpid openssl
  pacemaker psmisc python python-minimal python2.7 python2.7-minimal resource-agents sgml-base
  shared-mime-info xml-core
Suggested packages:
  dbus-x11 gawk-doc libsasl2-modules-otp libsasl2-modules-ldap libsasl2-modules-sql
  libsasl2-modules-gssapi-mit libsasl2-modules-gssapi-heimdal lm-sensors snmp-mibs-downloader
  python-doc python-tk python2.7-doc binutils binfmt-support sgml-base-doc debhelper
The following NEW packages will be installed:
  ca-certificates cluster-glue dbus file gawk heartbeat libcap2 libcfg4 libcib1 libconfdb4
  libcoroipcc4 libcoroipcs4 libcorosync4 libcpg4 libcrmcluster1 libcrmcommon2 libcurl3 libdbus-1-3
  libdbus-glib-1-2 libesmtp6 libevs4 libffi5 libglib2.0-0 libglib2.0-data libheartbeat2 libldap-2.4-2
  liblogsys4 liblrm2 libltdl7 libmagic1 libnet1 libnspr4 libnspr4-0d libnss3 libnss3-1d libopenhpi2
  libopenipmi0 libpcre3 libpe-rules2 libpe-status3 libpengine3 libperl5.14 libpils2 libpload4
  libplumb2 libplumbgpl2 libquorum4 librtmp0 libsam4 libsasl2-2 libsasl2-modules libsensors4
  libsigsegv2 libsnmp-base libsnmp15 libssh2-1 libstonith1 libstonithd1 libsystemd-login0
  libtimedate-perl libtotem-pg4 libtransitioner1 libvotequorum4 libxml2 libxml2-utils libxslt1.1
  mime-support openhpid openssl pacemaker psmisc python python-minimal python2.7 python2.7-minimal
  resource-agents sgml-base shared-mime-info xml-core
0 upgraded, 79 newly installed, 0 to remove and 0 not upgraded.
Need to get 25.7 MB of archives.
After this operation, 71.1 MB of additional disk space will be used.
Do you want to continue [Y/n]?

[ ... ]

Fetched 25.7 MB in 1min 17s (330 kB/s)
                                                                                          Extracting
    templates from packages: 100%
Preconfiguring packages ...

[ ... ]
```

Now `/etc/ha.d/ha.cf` can be created with this content:

```
logfacility daemon

node iscsi-server0
node iscsi-server1

keepalive 1
warntime 2
deadtime 3

udpport 694
ucast eth0 192.168.103.99
ucast eth0 192.168.103.91

auto_failback off
```

It says there will be two nodes named iscsi-server0 and iscsi-server1, they will communicate by unicast on eth0 interface to their private IP addresses, and there will be a keep-alive heartbeat every second. A warning will be reached on 2 second delay and at three seconds a fail-over will occur. So, it is configured a RTO of three seconds plus service startup time.

auto_failback off is used to avoid a new outage when primary node comes back again, secondary node keeps being active one.

Now /etc/ha.d/authkeys is configured to define authentication between nodes, in production systems *RandomSecretPassword* should be substituted by a real password:

```
auth 1
1 sha1 RandomSecretPassword
```

And finally /etc/haresources should contain the services and the initial node that will provide them:

```
iscsi-server0   192.168.1.36/24 tgt
```

If those three files are synced between servers, heartbeat can be started now, again on both of them:

```
# /etc/init.d/heartbeat start
Starting High-Availability services: IPaddr[10744]: INFO:  Resource is stopped
Done.
```

If looked at system logs, it will be seen what happened to the services:

```
Nov  2 20:10:46 iscsi-server0 heartbeat: [12849]: WARN: Core dumps could be lost if multiple dumps occur.
Nov  2 20:10:46 iscsi-server0 heartbeat: [12849]: WARN: Consider setting non-default value in /proc/sys/
     kernel/core_pattern (or equivalent) for maximum supportability
Nov  2 20:10:46 iscsi-server0 heartbeat: [12849]: WARN: Consider setting /proc/sys/kernel/core_uses_pid (or
     equivalent) to 1 for maximum supportability
Nov  2 20:10:46 iscsi-server0 heartbeat: [12849]: info: Pacemaker support: false
Nov  2 20:10:46 iscsi-server0 heartbeat: [12849]: WARN: Logging daemon is disabled --enabling logging daemon
      is recommended
Nov  2 20:10:46 iscsi-server0 heartbeat: [12849]: info: **************************
Nov  2 20:10:46 iscsi-server0 heartbeat: [12849]: info: Configuration validated. Starting heartbeat 3.0.5
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: heartbeat: version 3.0.5
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: Heartbeat generation: 1383418327
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: write socket priority set to
     IPTOS_LOWDELAY on eth0
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: bound send socket to device: eth0
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: bound receive socket to device: eth0
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: started on port 694 interface eth0 to
     192.168.103.99
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: write socket priority set to
     IPTOS_LOWDELAY on eth0
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: bound send socket to device: eth0
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: bound receive socket to device: eth0
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: glib: ucast: started on port 694 interface eth0 to
     192.168.103.91
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: Local status now set to: 'up'
Nov  2 20:10:46 iscsi-server0 heartbeat: [12850]: info: Link iscsi-server0:eth0 up.
Nov  2 20:10:52 iscsi-server0 heartbeat: [12850]: info: Link iscsi-server1:eth0 up.
Nov  2 20:10:52 iscsi-server0 heartbeat: [12850]: info: Status update for node iscsi-server1: status up
Nov  2 20:10:52 iscsi-server0 heartbeat: [12860]: debug: notify_world: setting SIGCHLD Handler to SIG_DFL
Nov  2 20:10:52 iscsi-server0 harc[12860]: info: Running /etc/ha.d//rc.d/status status
Nov  2 20:10:53 iscsi-server0 heartbeat: [12850]: debug: get_delnodelist: delnodelist=
Nov  2 20:10:53 iscsi-server0 heartbeat: [12850]: info: Comm_now_up(): updating status to active
Nov  2 20:10:53 iscsi-server0 heartbeat: [12850]: info: Local status now set to: 'active'
Nov  2 20:10:53 iscsi-server0 heartbeat: [12850]: info: Status update for node iscsi-server1: status active
Nov  2 20:10:53 iscsi-server0 heartbeat: [12878]: debug: notify_world: setting SIGCHLD Handler to SIG_DFL
Nov  2 20:10:53 iscsi-server0 harc[12878]: info: Running /etc/ha.d//rc.d/status status
Nov  2 20:11:03 iscsi-server0 heartbeat: [12850]: info: remote resource transition completed.
Nov  2 20:11:03 iscsi-server0 heartbeat: [12850]: info: remote resource transition completed.
Nov  2 20:11:03 iscsi-server0 heartbeat: [12850]: info: Initial resource acquisition complete (T_RESOURCES(
     us))
Nov  2 20:11:04 iscsi-server0 IPaddr[12936]: INFO:  Resource is stopped
Nov  2 20:11:04 iscsi-server0 heartbeat: [12901]: info: Local Resource acquisition completed.
Nov  2 20:11:04 iscsi-server0 heartbeat: [12850]: debug: StartNextRemoteRscReq(): child count 1
Nov  2 20:11:04 iscsi-server0 heartbeat: [12973]: debug: notify_world: setting SIGCHLD Handler to SIG_DFL
Nov  2 20:11:04 iscsi-server0 harc[12973]: info: Running /etc/ha.d//rc.d/ip-request-resp ip-request-resp
Nov  2 20:11:04 iscsi-server0 ip-request-resp[12973]: received ip-request-resp 192.168.103.36/24 OK yes
Nov  2 20:11:04 iscsi-server0 ResourceManager[12994]: info: Acquiring resource group: iscsi-server0
     192.168.103.36/24 tgt
Nov  2 20:11:04 iscsi-server0 IPaddr[13021]: INFO:  Resource is stopped
Nov  2 20:11:04 iscsi-server0 ResourceManager[12994]: info: Running /etc/ha.d/resource.d/IPaddr
     192.168.103.36/24 start
Nov  2 20:11:04 iscsi-server0 IPaddr[13097]: INFO: Using calculated nic for 192.168.103.36: eth0
Nov  2 20:11:04 iscsi-server0 IPaddr[13097]: INFO: Using calculated netmask for 192.168.103.36:
     255.255.255.0
Nov  2 20:11:04 iscsi-server0 IPaddr[13097]: INFO: eval ifconfig eth0:0 192.168.103.36 netmask 255.255.255.0
      broadcast 192.168.103.255
Nov  2 20:11:04 iscsi-server0 IPaddr[13076]: INFO:  Success
Nov  2 20:11:04 iscsi-server0 ResourceManager[12994]: info: Running /etc/init.d/tgt  start
Nov  2 20:11:05 iscsi-server0 tgtd: semkey 0x610e403c
Nov  2 20:11:05 iscsi-server0 tgtd: tgtd daemon started, pid:13216
Nov  2 20:11:05 iscsi-server0 tgtd: tgtd logger started, pid:13218 debug:0
Nov  2 20:11:05 iscsi-server0 tgtd: work_timer_start(146) use timer_fd based scheduler
Nov  2 20:11:05 iscsi-server0 tgtd: bs_init(312) use signalfd notification
```

And now if it is checked if services had been started, they can be seen, both IP address is set, and `tgt` is exporting out ISCSI target:

```
root@iscsi-server0:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 52:54:00:ba:cf:e6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.103.99/24 brd 192.168.103.255 scope global eth0
    inet 192.168.103.36/24 brd 192.168.103.255 scope global secondary eth0:0
    inet6 fe80::5054:ff:feba:cfe6/64 scope link
       valid_lft forever preferred_lft forever
root@iscsi-server0:~# tgtadm --lld iscsi --op show --mode target
Target 1: iqn.2013-11.edu.uoc:ha-disk
    System information:
        Driver: iscsi
        State: ready
    I_T nexus information:
    LUN information:
        LUN: 0
            Type: controller
            SCSI ID: IET     00010000
            SCSI SN: beaf10
            Size: 0 MB, Block size: 1
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: null
            Backing store path: None
            Backing store flags:
        LUN: 1
            Type: disk
            SCSI ID: IET     00010001
            SCSI SN: beaf11
            Size: 231 MB, Block size: 512
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: rdwr
            Backing store path: /dev/drbd0
            Backing store flags:
    Account information:
    ACL information:
        ALL
```

2.4.2.4 ISCSI client

Client should now connect to exported ISCSI target. From now on all work will be done on `iscsi-client`. ISCSI should be configured, needed client software is `open-iscsi`:

```
root@iscsi-client:~# apt-get install open-iscsi
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  open-iscsi
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 532 kB of archives.
After this operation, 2,166 kB of additional disk space will be used.
Get:1 http://ftp.fi.debian.org/debian/ wheezy/main open-iscsi amd64 2.0.873-3 [532 kB]
Fetched 532 kB in 2s (266 kB/s)
Selecting previously unselected package open-iscsi.
(Reading database ... 18431 files and directories currently installed.)
Unpacking open-iscsi (from .../open-iscsi_2.0.873-3_amd64.deb) ...
Processing triggers for man-db ...
Setting up open-iscsi (2.0.873-3) ...
```

To see all exported targets from `iscsi-server-ha` the service can be started and this command written:

```
root@iscsi-client:~# /etc/init.d/open-iscsi start
[ ok ] Starting iSCSI initiator service: iscsid.
[....] Setting up iSCSI targets:
iscsiadm: No records found
. ok
[ ok ] Mounting network filesystems:.
root@iscsi-client:~# iscsiadm --mode discovery --type sendtargets --portal iscsi-server-ha
192.168.103.36:3260,1 iqn.2013-11.edu.uoc:ha-disk
```

Now `iscsi-client` should connect to this target:

```
root@iscsi-client:~# iscsiadm --mode node --targetname iqn.2013-11.edu.uoc:ha-disk --portal iscsi-server-ha
      --login
Logging in to [iface: default, target: iqn.2013-11.edu.uoc:ha-disk, portal: 192.168.103.36,3260] (multiple)
Login to [iface: default, target: iqn.2013-11.edu.uoc:ha-disk, portal: 192.168.103.36,3260] successful.
```

Now a new disk (`/dev/sdb`) is attached to this system, looking at `/var/log/syslog` it can be seen:

```
Nov  2 20:41:14 iscsi-client kernel: [25432.581082] scsi2 : iSCSI Initiator over TCP/IP
Nov  2 20:41:15 iscsi-client kernel: [25433.256887] scsi 2:0:0:0: RAID              IET      Controller
           0001 PQ: 0 ANSI: 5
Nov  2 20:41:15 iscsi-client kernel: [25433.303447] scsi 2:0:0:0: Attached scsi generic sg2 type 12
Nov  2 20:41:15 iscsi-client kernel: [25433.305431] scsi 2:0:0:1: Direct-Access     IET      VIRTUAL-DISK
         0001 PQ: 0 ANSI: 5
Nov  2 20:41:15 iscsi-client kernel: [25433.309740] sd 2:0:0:1: Attached scsi generic sg3 type 0
Nov  2 20:41:15 iscsi-client kernel: [25433.317589] sd 2:0:0:1: [sdb] 450472 512-byte logical blocks: (230
    MB/219 MiB)
Nov  2 20:41:15 iscsi-client kernel: [25433.318455] sd 2:0:0:1: [sdb] Write Protect is off
Nov  2 20:41:15 iscsi-client kernel: [25433.318458] sd 2:0:0:1: [sdb] Mode Sense: 49 00 00 08
Nov  2 20:41:15 iscsi-client kernel: [25433.319215] sd 2:0:0:1: [sdb] Write cache: enabled, read cache:
     enabled, doesn't support DPO or FUA
Nov  2 20:41:15 iscsi-client kernel: [25433.408323]  sdb: unknown partition table
Nov  2 20:41:15 iscsi-client kernel: [25433.441373] sd 2:0:0:1: [sdb] Attached SCSI disk
Nov  2 20:41:15 iscsi-client iscsid: Connection1:0 to [target: iqn.2013-11.edu.uoc:ha-disk, portal:
     192.168.103.36,3260] through [iface: default] is operational now
```

`iscsi-client` can partition, format and mount it:

```
root@iscsi-client:~# fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x19dc69dd.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-450471, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-450471, default 450471):
Using default value 450471

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
root@iscsi-client:~# mkfs.ext4 /dev/sdb1
mke2fs 1.42.5 (29-Jul-2012)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
56224 inodes, 224212 blocks
11210 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
28 block groups
8192 blocks per group, 8192 fragments per group
2008 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729, 204801, 221185

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

root@iscsi-client:~# mount /dev/sdb1 /mnt/
root@iscsi-client:~# df -h
Filesystem                                              Size  Used Avail Use% Mounted on
rootfs                                                  3.8G  580M  3.0G  16% /
udev                                                     10M     0   10M   0% /dev
tmpfs                                                    50M  188K   50M   1% /run
/dev/disk/by-uuid/cab98ef2-c703-4e3e-883e-3c62ca200a97  3.8G  580M  3.0G  16% /
tmpfs                                                   5.0M     0  5.0M   0% /run/lock
tmpfs                                                   143M     0  143M   0% /run/shm
/dev/sdb1                                               213M  6.1M  196M   3% /mnt
root@iscsi-client:~# ls -l /mnt/
total 12
```

```
drwx------ 2 root root 12288 Nov  2 20:45 lost+found
```

And at ISCSI server the client connection can be seen:

```
root@iscsi-server0:~# tgtadm --lld iscsi --op show --mode target
Target 1: iqn.2013-11.edu.uoc:ha-disk
    System information:
        Driver: iscsi
        State: ready
    I_T nexus information:
        I_T nexus: 1
            Initiator: iqn.1993-08.org.debian:01:e4faf112bfb0
            Connection: 0
                IP Address: 192.168.103.93
    LUN information:
        LUN: 0
            Type: controller
            SCSI ID: IET     00010000
            SCSI SN: beaf10
            Size: 0 MB, Block size: 1
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: null
            Backing store path: None
            Backing store flags:
        LUN: 1
            Type: disk
            SCSI ID: IET     00010001
            SCSI SN: beaf11
            Size: 231 MB, Block size: 512
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: rdwr
            Backing store path: /dev/drbd0
            Backing store flags:
    Account information:
    ACL information:
        ALL
```

2.4.3 Failure test

Heartbeat should be shut down on `iscsi-server0` and `iscsi-server1` will takeover the services.

On `iscsi-server0`:

```
root@iscsi-server0:~# /etc/init.d/heartbeat stop
Stopping High-Availability services: Killed
```

And then on `iscsi-server1` the services are started, and `iscsi-client` is already connected:

```
root@iscsi-server1:/etc/ha.d# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 52:54:00:ce:65:76 brd ff:ff:ff:ff:ff:ff
    inet 192.168.103.91/24 brd 192.168.103.255 scope global eth0
    inet 192.168.103.36/24 brd 192.168.103.255 scope global secondary eth0:0
    inet6 fe80::5054:ff:fece:6576/64 scope link
       valid_lft forever preferred_lft forever
root@iscsi-server1:/etc/ha.d# tgtadm --lld iscsi --op show --mode target
Target 1: iqn.2013-11.edu.uoc:ha-disk
    System information:
        Driver: iscsi
        State: ready
    I_T nexus information:
        I_T nexus: 1
            Initiator: iqn.1993-08.org.debian:01:e4faf112bfb0
            Connection: 0
                IP Address: 192.168.103.93
    LUN information:
        LUN: 0
            Type: controller
            SCSI ID: IET     00010000
            SCSI SN: beaf10
            Size: 0 MB, Block size: 1
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: null
            Backing store path: None
            Backing store flags:
        LUN: 1
            Type: disk
            SCSI ID: IET     00010001
            SCSI SN: beaf11
            Size: 231 MB, Block size: 512
            Online: Yes
            Removable media: No
            Readonly: No
            Backing store type: rdwr
            Backing store path: /dev/drbd0
            Backing store flags:
    Account information:
    ACL information:
        ALL
```

On `iscsi-client` those system logs can be seen:

```
Nov  2 20:54:05 iscsi-client iscsid: Kernel reported iSCSI connection 1:0 error (1020 -
     ISCSI_ERR_TCP_CONN_CLOSE: TCP connection closed) state (3)
Nov  2 20:54:08 iscsi-client iscsid: connection1:0 is operational after recovery (1 attempts)
```

So, just a 3 seconds downtime was suffered. It is fair to say heartbeat was stopped correctly on `iscsi-server0`, in case of power failure, detection would last 3 seconds, there will be an startup of a pair more seconds and client will not see a TCP FIN, instead it will see the server host (the new active one) says TCP connection doesn't exist and a longer downtime will occur.

This scenario should be tried. At laboratory, a brute-force shutting down of active ISCSI server (`iscsi-server1`) gives this result:

`iscsi-server0` detects the node has fallen down and starts services in two seconds:

```
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: WARN: node iscsi-server1: is dead
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: WARN: No STONITH device configured.
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: WARN: Shared disks are not protected.
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: info: Resources being acquired from iscsi-server1.
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: info: Link iscsi-server1:eth0 dead.
Nov  2 21:15:15 iscsi-server0 heartbeat: [2336]: debug: notify_world: setting SIGCHLD Handler to SIG_DFL
Nov  2 21:15:15 iscsi-server0 harc[2336]: info: Running /etc/ha.d//rc.d/status status
Nov  2 21:15:15 iscsi-server0 mach_down[2370]: info: /usr/share/heartbeat/mach_down: nice_failback: foreign
     resources acquired
```

```
Nov  2 21:15:15 iscsi-server0 mach_down[2370]: info: mach_down takeover complete for node iscsi-server1.
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: info: mach_down takeover complete.
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: debug: StartNextRemoteRscReq(): child count 1
Nov  2 21:15:15 iscsi-server0 IPaddr[2394]: INFO:  Resource is stopped
Nov  2 21:15:15 iscsi-server0 heartbeat: [2337]: info: Local Resource acquisition completed.
Nov  2 21:15:15 iscsi-server0 heartbeat: [2147]: debug: StartNextRemoteRscReq(): child count 1
Nov  2 21:15:15 iscsi-server0 heartbeat: [2457]: debug: notify_world: setting SIGCHLD Handler to SIG_DFL
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.688587] block drbd0: PingAck did not arrive in time.
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.688925] block drbd0: peer( Primary -> Unknown ) conn( Connected
      -> NetworkFailure ) pdsk( UpToDate -> DUnknown )
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.688990] block drbd0: new current UUID 8241221A753008EF:209
     F54EACEE19147:707A82C51BBD9A73:707982C51BBD9A73
Nov  2 21:15:15 iscsi-server0 harc[2457]: info: Running /etc/ha.d//rc.d/ip-request-resp ip-request-resp
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.705233] block drbd0: asender terminated
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.705237] block drbd0: Terminating drbd0_asender
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.713601] block drbd0: Connection closed
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.713608] block drbd0: conn( NetworkFailure -> Unconnected )
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.713611] block drbd0: receiver terminated
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.713613] block drbd0: Restarting drbd0_receiver
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.713615] block drbd0: receiver (re)started
Nov  2 21:15:15 iscsi-server0 kernel: [ 1265.713618] block drbd0: conn( Unconnected -> WFConnection )
Nov  2 21:15:15 iscsi-server0 ip-request-resp[2457]: received ip-request-resp 192.168.103.36/24 OK yes
Nov  2 21:15:15 iscsi-server0 ResourceManager[2478]: info: Acquiring resource group: iscsi-server0
     192.168.103.36/24 tgt
Nov  2 21:15:15 iscsi-server0 IPaddr[2505]: INFO:  Resource is stopped
Nov  2 21:15:15 iscsi-server0 ResourceManager[2478]: info: Running /etc/ha.d/resource.d/IPaddr
     192.168.103.36/24 start
Nov  2 21:15:16 iscsi-server0 IPaddr[2581]: INFO: Using calculated nic for 192.168.103.36: eth0
Nov  2 21:15:16 iscsi-server0 IPaddr[2581]: INFO: Using calculated netmask for 192.168.103.36: 255.255.255.0
Nov  2 21:15:16 iscsi-server0 IPaddr[2581]: INFO: eval ifconfig eth0:0 192.168.103.36 netmask 255.255.255.0
     broadcast 192.168.103.255
Nov  2 21:15:16 iscsi-server0 IPaddr[2560]: INFO:  Success
Nov  2 21:15:16 iscsi-server0 ResourceManager[2478]: info: Running /etc/init.d/tgt  start
Nov  2 21:15:16 iscsi-server0 kernel: [ 1266.797245] tgtd (2700): /proc/2700/oom_adj is deprecated, please
     use /proc/2700/oom_score_adj instead.
Nov  2 21:15:16 iscsi-server0 tgtd: semkey 0x610e17ce
Nov  2 21:15:16 iscsi-server0 tgtd: tgtd daemon started, pid:2700
Nov  2 21:15:16 iscsi-server0 tgtd: tgtd logger started, pid:2702 debug:0
Nov  2 21:15:17 iscsi-server0 tgtd: work_timer_start(146) use timer_fd based scheduler
Nov  2 21:15:17 iscsi-server0 tgtd: bs_init(312) use signalfd notification
```

`iscsi-client` detects the outage when service is reestablished and then reconnects in three seconds:

```
Nov  2 21:15:17 iscsi-client iscsid: Kernel reported iSCSI connection 1:0 error (1020 -
     ISCSI_ERR_TCP_CONN_CLOSE: TCP connection closed) state (3)
Nov  2 21:15:20 iscsi-client iscsid: connection1:0 is operational after recovery (1 attempts)
```

In this case a downtime of a total of eight seconds is reached:

- 3 seconds for heartbeat detection
- 2 seconds for service restoration
- 3 seconds for client reconnection

2.4.4 Possible improvements

These numbers can be improved in real production systems:

- heartbeats can be sent more often than once per second, and detection can be done in less than a second.

- tgt service can be always up, it's a bit dangerous because no clients should connect no normal IP address, this can be assured by other means like iptables, so finally service restoration can be achieved in the time an IP address is assigned, current IP assignment is done by `/etc/ha.d/resources.d/IPaddr` script. This script can also be optimized.
- client reconnection will be more difficult to improve as it depends on `iscsid` application and Network connectivity. A faster connection may help and may be ISCSI client software can be substituted, also other techniques can be used. ISCSI multi-path permits a system to maintain more than one *path* to the same target. No reconnection is needed to put or retrieve data, as other paths are available. See figure 2.3 for a graphical explanation.

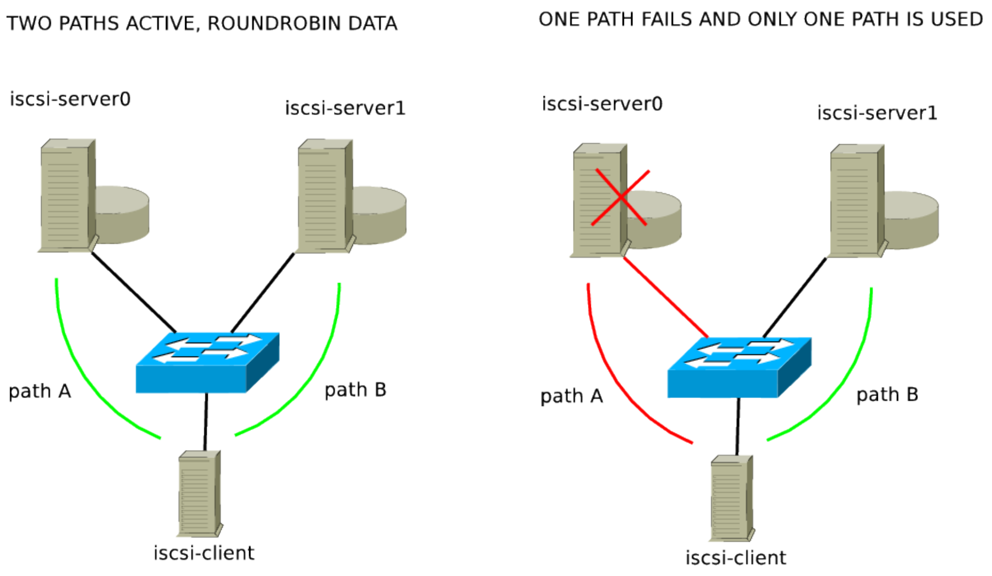


Figure 2.3: Multi-path

Chapter 3

Networking

3.1 Ways to provide high available networking

For two hosts to communicate, there are many possible points of failure. Usually vendors, for example Sun Microsystems[40] and Cisco[5] classify these points in three categories or levels:

- Link level: used to provide Network Interface Card or Switch redundancy.
- Gateway level: used to provide Gateway router redundancy.
- Routing level: used to provide topology failure redundancy.

Figure 3.1 shows a diagram with all these levels combined. Normally all of them should be used to provide a *no single point of failure* redundancy.

Let's explain each one in detail in next sections.

3.1.1 Link Level

A network card (NIC) may fail, so a host should have more than one card to provide connectivity in case this happens. Also a switch may fail, so it's a good practice to have more than one switch for the same network, and hosts connected to more than one of them, so in case of failure there is another switch that can carry on with connectivity. This is named *Link Aggregation*.

So, at least two switches are needed and all high available hosts should have a minimum of two interface cards.

There are also many ways to provide link aggregation, some of them only provide fail over, some others also provide higher throughput than single cards.

Those are the most commonly used link aggregations methods that provide network redundancy:

- Active/passive: one host has one active card connected to one switch, is case this switch powers down or network card fails, the host promotes another card to active one. Normally the way to trigger fail over is to detect link failure. Also ARP queries may be used. All network cards are

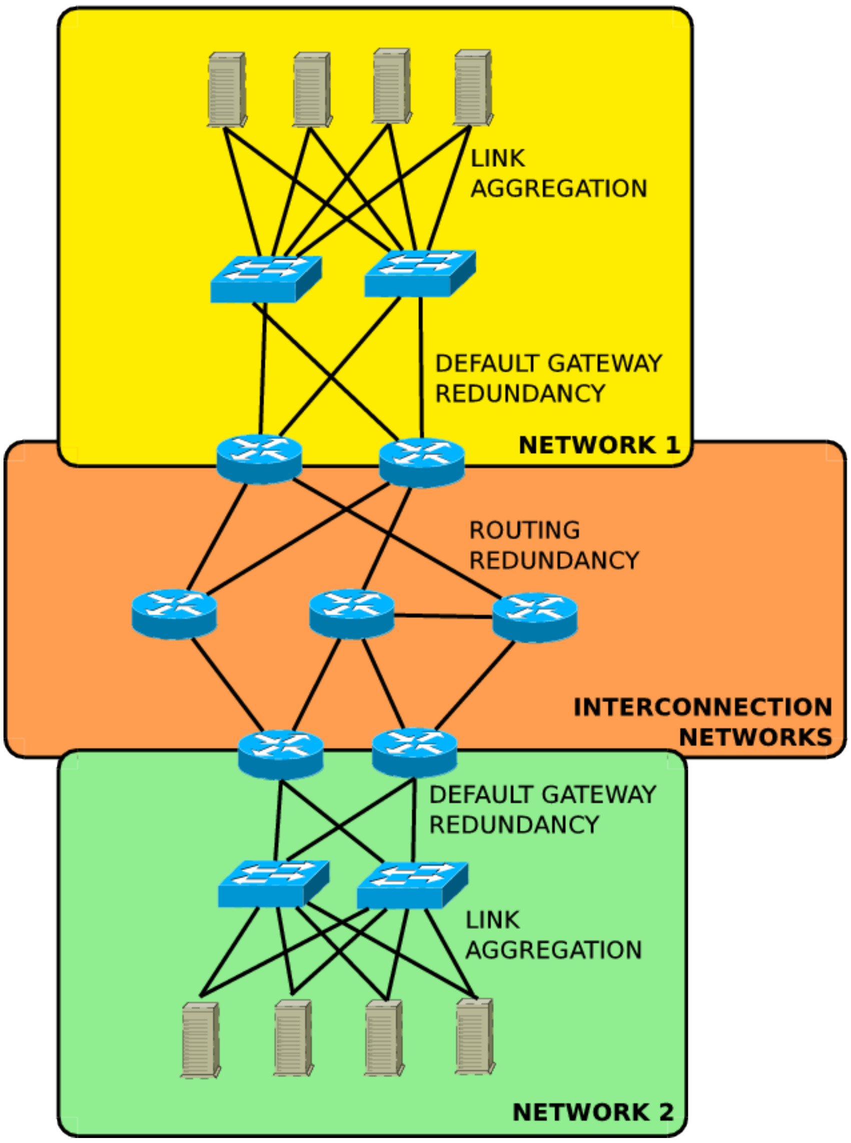


Figure 3.1: Network Redundancy

presented with a single MAC address. Switches should be connected between each other at layer 2 level because some hosts may use one as active and others, another one. Some hosts may not have more than one NIC and act as a bonding with a single active card. No special Switch capability is needed.

- LACP[17]: one host has more than one card connected to the same host. They speak 802.3ad protocol to sum bandwidth of all network cards. With this method also all network cards are presented with the same MAC address, each Ethernet packet is sent using round robin through all NICs. This method increases throughput, but only uses one switch, so switch failure is not supported. Switch capability is needed two support this method.

- Adaptive Load Balancing[9]: one host has more than one card and each one connected to a different switch. All are active at once. To share bandwidth when a host receives a ARP request, it replies with the MAC associated to least used card. So all cards have different MAC address. No special switch capability is needed. This is may be the preferred way to provide network redundancy as increases throughput and provides full redundancy.

At figure 3.1 we can see an example link aggregation redundancy network on both networks.

3.1.2 Gateway Level

All hosts may have multiple NICs to connect each other at a layer 2 network. But all of them will have a default gateway to connect to other hosts outside their network. The default gateway router may also have multiple NICs, but the whole host may fail so it is needed a way to provide more than one default router transparent to network hosts.

This redundancy can be in active/active mode but a lot of work will be need in every host to support it, so normally active/passive systems are used. A Virtual IP address is shared by all routers an only active one uses it and replies to ARP requests using that IP address. A protocol is used to talk between potential routers. There is a standard named VRRP[59] if different vendors want to be used. But normally all routers are from the same vendor and proprietary protocols are used. In GNU/Linux Heartbeat[23] is an application that can be used to have this Virtual IP address, and BSD systems use one software named CARP[45].

Also, there can be techniques to share network sessions to be able to fail over a firewall without disrupting active sessions.

At figure 3.1 both network border routers represent the system explained in this section

3.1.3 Routing level

For bigger interconnected networks Gateway Level methods are not used because of complexity. Also there are a lot of network routes that should be configured and this routes, in case of topology failure, should be reconfigured. So it is needed a way to propagate network routes between a group of routers. When all routes have arrived to all routers and have been applied it is said that they have reach convergence.

There are may routing protocols[55]. Internet uses BGP[43] that doesn't route routes but *Autonomous Systems*, because there are so many routes that won't be practical to route them individually at large scale. Inside an Autonomous System other protocols[55] are used, they are named Interior Border Protocols. For fast convergence, in case of network failure, preferred protocol is OSPF[53]. But all of them provide the same functionality. Each router publishes known routes to foreign routers and stores best gateways to reach not directly connected networks.

Figure 3.1 shows an example routing connection between seven routers. On convergence, all routers will have IP routes defined to use gateways that get the shortest path to reach them.

3.2 The GNU/Linux way

GNU/Linux owns open source solutions for all these network redundancy targets.

3.2.1 Link level

In vanilla Linux kernel, the `bonding` driver[9] supports all mentioned levels and some others. They are numbered from 1 to 6. There are two ways to configure it:

- By loading the `bonding` driver with `mode=XX` parameter. This is only suited for single bonding devices in a host.
- By loading the driver and interacting with `/sys/class/net/bonding_masters` file and `/sys/class/net/bondX` folders. This is the preferred method for hosts connected to multiple networks, as more than one bonding device and different modes can be used. `cat` and `echo` can be used to add/remove or modify configuration. Debian and Redhat/CentOS network parameters at `/etc/network/interfaces` and `/etc/sysconfig/network/ifcfg-bondX` respectively uses this method.

Some modes will require that switches are connected, some others will require the contrary.

We will explain all modes in detail in next subsections.

3.2.1.1 Balance round robin (mode=0)

> *Round-robin policy: Transmit packets in sequential order from the first available slave through the last.*[9]

Each single packet uses a network device. For outgoing traffic it is balanced, for incoming traffic switches will have cached the MAC address in a particular port.

It is needed to connect all switches between them because a host many have a failing device to one switch. May use a lot of bandwidth between switches.

3.2.1.2 Active backup (mode=1)

> *Active-backup policy: Only one slave in the bond is active. A different slave becomes active if, and only if, the active slave fails*.[9]

One active card, in case of failure another one is promoted to active.

Switches should be connected between each other.

3.2.1.3 Balance XOR (mode=2)

> *XOR policy: Transmit based on the selected transmit hash policy. The default policy is a simple [(source MAC address XOR'd with destination MAC address) modulo slave count].*[9]

Destination hosts are balanced depending on MAC address. It's a kind of sharing of destination hosts between network devices. As gateway has only one MAC address, default gateway will always use just one card.

Switches should also be connected.

3.2.1.4 Broadcast (mode=3)

> *Broadcast policy: transmits everything on all slave interfaces.*[9]

Repeated through all network interfaces, packets will be received duplicated.

It is better to not connect switches between each other. A lot of network traffic. Redundancy is provided because each host sends traffic to all switches.

3.2.1.5 802.3ad (mode=4)

> *IEEE 802.3ad Dynamic link aggregation. Creates aggregation groups that share the same speed and duplex settings. Utilizes all slaves in the active aggregator according to the 802.3ad specification.* [9]

Standard LACP[17], all ports should be connected to the same switch. That need to support LACP and has to be configured to define a group with all ports this host is connected to.

3.2.1.6 Balance-tlb (mode=5)

> *Adaptive transmit load balancing: channel bonding that does not require any special switch support. The outgoing traffic is distributed according to the current load (computed relative to the speed) on each slave. Incoming traffic is received by the current slave. If the receiving slave fails, another slave takes over the MAC address of the failed receiving slave.* [9]

All NICs are active, outgoing traffic is shared between devices depending on usage. Incoming traffic is received on a single card.

Switches should be connected.

3.2.1.7 Balance-alb (mode=6)

> *Adaptive load balancing: includes balance-tlb plus receive load balancing (rlb) for IPV4 traffic, and does not require any special switch support.*[9]

Uses the same techniques as balance-tld but also shares incoming traffic answering different mac addresses to ARP requests, so distributes both outgoing and incoming traffic between slaves.

Switches should be connected.

3.2.2 Gateway Level

There are many applications that can be used to know if other hosts are alive, in order to launch an IP address assignment in case of active router failure.

VRRP is supported in GNU/Linux [18], but normally other applications are used. In subsection 2.4.2.3 we configured Heartbeat[23] to have a Virtual IP address for active system. The same configuration could be used to define default gateway IP address between two or more routers. Also Keepalived[21] can be used.

If routers also have firewall capabilities it would be a good idea to have sessions in sync. For that an application named conntrackd[26] exists, installed on both routers is used to notify passive router with new sessions. Combined with heartbeat, for example, can be used to populate kernel with all sessions received from the previously active router.

3.2.3 Routing Level

Some routing protocol applications exist. The most long used one may be Quagga [36] and is also the one that we will use in our practical example. Lately BIRD [41] is having a lot of acceptance as it seems to scale better. Both of them support many routing protocols[55].

Quagga has a user-friendly CLI for people used to Cisco interfaces as it has nearly the same syntax. It supports BGP4, BGP4+, OSPFv2, OSPFv3, IS-IS, RIPv1, RIPv2, and RIPng.

Bird has fewer supported protocols, but as said before, it seems to have also fewer CPU needs[35]. It supports OSPFv2, OSPFv3, RIPv2, RIPng and BGP.

3.3 Practical Example

3.3.1 Description

In this example, a Layer-3 high available network will be built. There will be two networks composed by two equipments: a host simulating a PC and a router. The routers of both networks will have two paths to connect each other. There will be two additional routers, one named `pathA` and other one named `pathB`, and both will have two network devices connected to the routers of the two previous networks.

The idea is to have high available paths to connect between both networks. Figure 3.2 shows the disposition of all routers and hosts. `router1` and `router2` will have the possibility to use the path that contains `pathA` or `pathB` router.

In order to build the laboratory, all six hosts will be Debian GNU/Linux virtual machines that will have as many Ethernet devices as show on figure 3.2, but will have an additional (`eth0`) network device to connect to Internet. The host that will storage those guests will have seven GNU/Linux bridges to include the virtual devices of all hosts. Figure 3.3 shows each host network devices, which bridge they are attached to, and IP addresses.

Routing software will be Quagga with OSPF module enabled. It will be configured with very frequent messages, so a router failure will be quickly detected and routes will also be quickly reorganized.

To test failure, `pathA` and `pathB` routers will be shut down and route propagation will be seen to get the high available network. ICMP echo request will be sending to see how many replies are lost between failures.

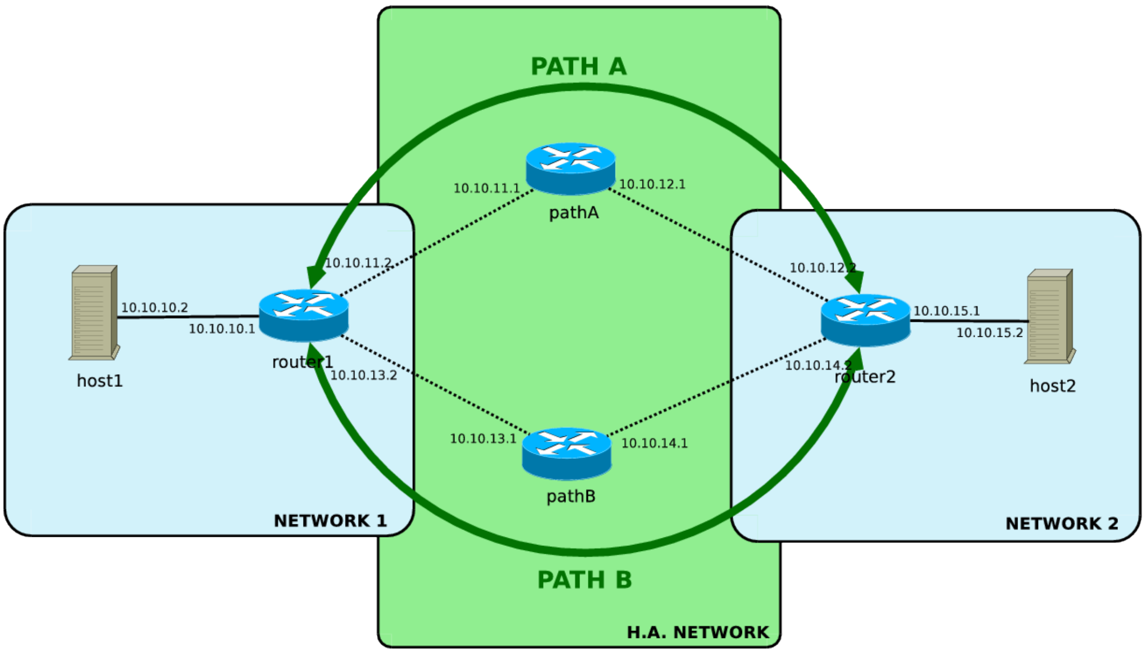


Figure 3.2: OSPF dual-path network

3.3.2 Configuration

3.3.2.1 `host1` and `host2`

Those systems will have `eth0` network device already configured to have Internet access. `eth1` device will be configured now.

`/etc/network/interfaces` can be edited on `host1` to add `eth1` declaration as follows:

```
auto eth1
iface eth1 inet static
        address 10.10.10.2
        netmask 255.255.255.0
        up ip route add 10.10.0.0/16 via 10.10.10.1
```

For `host2` IP address will be 10.10.15.2. And that's it. No more configuration is needed on `host1` and `host2`. They are supposed to be just PCs with a single gateway. In this example 10.10.0.0/16 network is routed by the gateway, which includes all networks in our local test. Internet is reached by `eth0` interface and it is not declared here.

3.3.2.2 `router1` and `router2`

First, is is needed to install Quagga. All IP addresses will be configured there.

```
# apt-get install quagga
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

	br0	br10	br11	br12	br13	br14	br15
HOST	eth0						
host1	eth0	eth1 10.10.10.2					
router1	eth0	eth1 10.10.10.1	eth2 10.10.11.2		eth3 10.10.13.2		
pathA	eth0		eth1 10.10.11.1	eth2 10.10.12.1			
pathB	eth0				eth1 10.10.13.1	eth2 10.10.14.1	
router2	eth0			eth1 10.10.12.2		eth2 10.10.14.2	eth3 10.10.15.1
host2	eth0						eth1 10.10.15.2

Figure 3.3: hosts, bridges and Ethernet devices

```
The following extra packages will be installed:
  libcap2
Suggested packages:
  snmpd
The following NEW packages will be installed:
  libcap2 quagga
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,722 kB of archives.
After this operation, 5,896 kB of additional disk space will be used.
Do you want to continue [Y/n]?
Get:1 http://ftp.fi.debian.org/debian/ wheezy/main libcap2 amd64 1:2.22-1.2
     [13.6 kB]
Get:2 http://ftp.fi.debian.org/debian/ wheezy/main quagga amd64 0.99.21-4+
   wheezy1 [1,708 kB]
Fetched 1,722 kB in 20s (82.7 kB/s)
Preconfiguring packages ...
Selecting previously unselected package libcap2:amd64.
(Reading database ... 18982 files and directories currently installed.)
Unpacking libcap2:amd64 (from .../libcap2_1%3a2.22-1.2_amd64.deb) ...
Selecting previously unselected package quagga.
Unpacking quagga (from .../quagga_0.99.21-4+wheezy1_amd64.deb) ...
Processing triggers for man-db ...
Setting up libcap2:amd64 (1:2.22-1.2) ...
Setting up quagga (0.99.21-4+wheezy1) ...
Loading capability module if not yet done.
Starting Quagga daemons (prio:10):.
```

Later `/etc/quagga/daemons` is edited and both `zebra` and `ospfd` daemons are enabled:

- `zebra` will be responsible to talk to the Linux kernel to apply route changes negotiated with foreign routers.

- `ospfd` will be responsible to talk to other OSPF daemons in the network to publish and receive routes and network changes

So, it will be left with this content:

```
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
isisd=no
babeld=no
```

At `quagga`, all daemons listen in a different TCP port for configuration. `vtysh` is a command that unifies all those daemons in a single administration prompt. `quagga` configuration follows the same language that Cisco IOS does, so people used to this language will feel comfortable. Indeed almost all routers follow the same language structure, `quagga` is not an exception.

Now `/etc/quagga/vtysh.conf` is edited:

```
hostname router1
username root nopassword
```

For `router2` host name is changed to `router2`.

Now `/etc/quagga/zebra.conf` of `router1` is edited:

```
interface eth1
 description link to host1
 link-detect
 ip address 10.10.10.1/24
 ipv6 nd suppress-ra
!
interface eth2
 description link to pathA
 link-detect
 ip address 10.10.11.2/24
 ipv6 nd suppress-ra
!
interface eth3
 description link to pathB
 link-detect
 ip address 10.10.13.2/24
 ipv6 nd suppress-ra
!
interface lo
!
```

```
ip forwarding
!
!
line vty
!
```

For `router2` it is just needed to change IP addresses and descriptions using the ones described at figure 3.3.

Then `/etc/quagga/ospfd.conf` needs to be edited, this is the content for `router1`:

```
!
!
!
!
!
interface eth1
 description link to host1
 ip ospf authentication null
 ip ospf hello-interval 1
 ip ospf dead-interval minimal hello-multiplier 5
!
interface eth2
 description link to pathA
 ip ospf authentication null
 ip ospf hello-interval 1
 ip ospf dead-interval minimal hello-multiplier 5
!
interface eth3
 description link to pathB
 ip ospf authentication null
 ip ospf hello-interval 1
 ip ospf dead-interval minimal hello-multiplier 5
!
interface lo
!
router ospf
 ospf router-id 192.168.103.80
 redistribute connected
 network 10.10.10.0/24 area 0.0.0.0
 network 10.10.11.0/24 area 0.0.0.0
 network 10.10.13.0/24 area 0.0.0.0
!
line vty
!
```

For fast convergence (High Availability when talking about network routes), important configurations are:

```
 ip ospf hello-interval 1
 ip ospf dead-interval minimal hello-multiplier 5
```

This means that failure detection should occur in less than a second. Hello messages will be sent at a ratio of 5 per second (every 200 milliseconds).

For `router2` interface descriptions need to be changed following figure 3.3, and inside `router ospf` section router-id show be different, in this case, it is `eth0` IP address. Also, in the same sections, all networks need to be declared to enable OSPF on each router.

Now `quagga` can be restarted:

```
# /etc/init.d/quagga restart
Stopping Quagga monitor daemon: (watchquagga).
Stopping Quagga daemons (prio:0): (ospfd) (zebra) (bgpd) (ripd) (ripngd) (
   ospf6d) (isisd) (babeld).
Removing all routes made by zebra.
Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfd.
Starting Quagga monitor daemon: watchquagga.
```

3.3.2.3 **`pathA`** and **`pathB`**

Those last routers also need to have `quagga` installed. `/etc/quagga/daemons` and `/etc/quagga/vtysh.conf` have the same content as `router1` and `router2`.

For `/etc/quagga/zebra.conf` the content is similar, it is just needed to include IP addresses, networks and descriptions of figure 3.3. For example, for `pathA` router this is its content:

```
!
!
interface eth0
 ipv6 nd suppress-ra
!
interface eth1
 description link to router1
 link-detect
 ip address 10.10.11.1/24
 ipv6 nd suppress-ra
!
interface eth2
 description link to router2
 link-detect
 ip address 10.10.12.1/24
 ipv6 nd suppress-ra
```

```
!
interface lo
!
ip forwarding
!
!
line vty
```

The same for `/etc/quagga/ospfd.conf`, this is the content for `pathB`:

```
!
!
!
!
!
interface eth0
!
interface eth1
 description link to router1
 ip ospf authentication null
 ip ospf hello-interval 1
 ip ospf dead-interval minimal hello-multiplier 5
!
interface eth2
 description link to router2
 ip ospf authentication null
 ip ospf hello-interval 1
 ip ospf dead-interval minimal hello-multiplier 5
!
interface lo
!
router ospf
 ospf router-id 192.168.103.92
 redistribute connected
 network 10.10.11.0/24 area 0.0.0.0
 network 10.10.12.0/24 area 0.0.0.0
!
line vty
!
```

Now it is need to have a look at `vtysh` and shell prompts, this is `router1`:

```
root@router1:~# vtysh

Hello, this is Quagga (version 0.99.21).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
```

```
router1# show ip ospf  neighbor

    Neighbor ID Pri State           Dead Time Address         Interface
                  RXmtL RqstL DBsmL
192.168.103.92    1 Full/Backup        0.953s 10.10.11.1      eth2
   :10.10.11.2          0     0     0
192.168.103.98    1 Full/Backup        0.954s 10.10.13.1      eth3
   :10.10.13.2          0     0     0
router1# show ip ospf  route
============ OSPF network routing table ============
N    10.10.10.0/24         [10] area: 0.0.0.0
                           directly attached to eth1
N    10.10.11.0/24         [10] area: 0.0.0.0
                           directly attached to eth2
N    10.10.12.0/24         [20] area: 0.0.0.0
                           via 10.10.11.1, eth2
N    10.10.13.0/24         [10] area: 0.0.0.0
                           directly attached to eth3
N    10.10.14.0/24         [20] area: 0.0.0.0
                           via 10.10.13.1, eth3
N    10.10.15.0/24         [30] area: 0.0.0.0
                           via 10.10.11.1, eth2
                           via 10.10.13.1, eth3

============ OSPF router routing table =============
R    192.168.103.92        [10] area: 0.0.0.0, ASBR
                           via 10.10.11.1, eth2
R    192.168.103.98        [10] area: 0.0.0.0, ASBR
                           via 10.10.13.1, eth3
R    192.168.103.99        [20] area: 0.0.0.0, ASBR
                           via 10.10.11.1, eth2
                           via 10.10.13.1, eth3

============ OSPF external routing table ===========
N E2 192.168.103.0/24      [10/20] tag: 0
                           via 10.10.11.1, eth2
                           via 10.10.13.1, eth3

router1# exit
root@router1:~# ip r
default via 192.168.103.1 dev eth0
10.10.10.0/24 dev eth1  proto kernel  scope link  src 10.10.10.1
10.10.11.0/24 dev eth2  proto kernel  scope link  src 10.10.11.2
10.10.12.0/24 via 10.10.11.1 dev eth2  proto zebra  metric 20
```

```
10.10.13.0/24 dev eth3  proto kernel  scope link  src 10.10.13.2
10.10.14.0/24 via 10.10.13.1 dev eth3  proto zebra  metric 20
10.10.15.0/24  proto zebra  metric 30
        nexthop via 10.10.11.1  dev eth2 weight 1
        nexthop via 10.10.13.1  dev eth3 weight 1
192.168.103.0/24 dev eth0  proto kernel  scope link  src 192.168.103.80
```

There are to paths with equal cost to reach 10.10.15.0/24 network, so zebra has configured *Equal Cost Multi-path* using both `router1` and `router2`, as expected. All other networks are also detected.

Now the same should be done for `pathB`:

```
root@pathB:~# vtysh

Hello, this is Quagga (version 0.99.21).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

pathB# show  ip ospf  neighbor

    Neighbor ID Pri State           Dead Time Address         Interface
                  RXmtL RqstL DBsmL
192.168.103.80    1 Full/DR           0.878s 10.10.13.2      eth1
   :10.10.13.1          0     0     0
192.168.103.99    1 Full/DR           0.878s 10.10.14.2      eth2
   :10.10.14.1          0     0     0
pathB# show ip ospf  route
============ OSPF network routing table ============
N    10.10.10.0/24         [20] area: 0.0.0.0
                           via 10.10.13.2, eth1
N    10.10.11.0/24         [20] area: 0.0.0.0
                           via 10.10.13.2, eth1
N    10.10.12.0/24         [20] area: 0.0.0.0
                           via 10.10.14.2, eth2
N    10.10.13.0/24         [10] area: 0.0.0.0
                           directly attached to eth1
N    10.10.14.0/24         [10] area: 0.0.0.0
                           directly attached to eth2
N    10.10.15.0/24         [20] area: 0.0.0.0
                           via 10.10.14.2, eth2

============ OSPF router routing table =============
R    192.168.103.80        [10] area: 0.0.0.0, ASBR
                           via 10.10.13.2, eth1
R    192.168.103.92        [20] area: 0.0.0.0, ASBR
                           via 10.10.13.2, eth1
                           via 10.10.14.2, eth2
```

```
R    192.168.103.99         [10] area: 0.0.0.0, ASBR
                            via 10.10.14.2, eth2

============ OSPF external routing table ===========
N E2 192.168.103.0/24       [10/20] tag: 0
                            via 10.10.13.2, eth1
                            via 10.10.14.2, eth2

pathB# exit
root@pathB:~# ip r
default via 192.168.103.1 dev eth0
10.10.10.0/24 via 10.10.13.2 dev eth1  proto zebra  metric 20
10.10.11.0/24 via 10.10.13.2 dev eth1  proto zebra  metric 20
10.10.12.0/24 via 10.10.14.2 dev eth2  proto zebra  metric 20
10.10.13.0/24 dev eth1  proto kernel  scope link  src 10.10.13.1
10.10.14.0/24 dev eth2  proto kernel  scope link  src 10.10.14.1
10.10.15.0/24 via 10.10.14.2 dev eth2  proto zebra  metric 20
192.168.103.0/24 dev eth0  proto kernel  scope link  src 192.168.103.98
```

And now that routes seem to have converged in all routers, let's try High Availability in next section.

3.3.3 Failure test

`host1` will keep sending a ping to `host2`, first `pathA` will be shut down and it will be seen what happens. Routes should be set to send all traffic through pathB. And there will be shown how many ICMP packets are lost.

Then `pathA` will be restored and routes should be reordered again. There should be few ICMP packets lost at each network change.

First, `host1` will ping `host2`:

```
root@host1:~# ping 10.10.15.2
PING 10.10.15.2 (10.10.15.2) 56(84) bytes of data.
64 bytes from 10.10.15.2: icmp_req=1 ttl=61 time=1.04 ms
64 bytes from 10.10.15.2: icmp_req=2 ttl=61 time=0.921 ms
64 bytes from 10.10.15.2: icmp_req=3 ttl=61 time=0.870 ms
64 bytes from 10.10.15.2: icmp_req=4 ttl=61 time=0.909 ms
```

Then `quagga` is stopped at `pathA`:

```
root@pathA:~# /etc/init.d/quagga stop
Stopping Quagga monitor daemon: watchquagga.
Stopping Quagga daemons (prio:0): ospfd zebra (bgpd) (ripd) (ripngd) (
   ospf6d) (isisd) (babeld).
Removing all routes made by zebra.
```

How many packets have been lost?

```
64 bytes from 10.10.15.2: icmp_req=10 ttl=61 time=0.835 ms
64 bytes from 10.10.15.2: icmp_req=11 ttl=61 time=0.963 ms
64 bytes from 10.10.15.2: icmp_req=12 ttl=61 time=0.885 ms
64 bytes from 10.10.15.2: icmp_req=13 ttl=61 time=0.821 ms
64 bytes from 10.10.15.2: icmp_req=14 ttl=61 time=0.964 ms
64 bytes from 10.10.15.2: icmp_req=16 ttl=61 time=0.915 ms
64 bytes from 10.10.15.2: icmp_req=17 ttl=61 time=0.980 ms
```

Just one, the one identified by sequence number 15.

Now a new pings will be sent, `quagga` will be started at `pathA`, convergence will be reached and `pathB` stopped.

First, `router1` shows that the only active path to `host2` is thought `pathB`:

```
root@router1:~# ip r
default via 192.168.103.1 dev eth0
10.10.10.0/24 dev eth1  proto kernel  scope link  src 10.10.10.1
10.10.11.0/24 dev eth2  proto kernel  scope link  src 10.10.11.2
10.10.12.0/24 via 10.10.13.1 dev eth3  proto zebra  metric 30
10.10.13.0/24 dev eth3  proto kernel  scope link  src 10.10.13.2
10.10.14.0/24 via 10.10.13.1 dev eth3  proto zebra  metric 20
10.10.15.0/24 via 10.10.13.1 dev eth3  proto zebra  metric 30
192.168.103.0/24 dev eth0  proto kernel  scope link  src 192.168.103.80
```

New ICMP packets will be sent with this topology:

```
root@host1:~# ping 10.10.15.2
PING 10.10.15.2 (10.10.15.2) 56(84) bytes of data.
64 bytes from 10.10.15.2: icmp_req=1 ttl=61 time=1.17 ms
64 bytes from 10.10.15.2: icmp_req=2 ttl=61 time=1.21 ms
64 bytes from 10.10.15.2: icmp_req=3 ttl=61 time=1.14 ms
64 bytes from 10.10.15.2: icmp_req=4 ttl=61 time=1.68 ms
64 bytes from 10.10.15.2: icmp_req=5 ttl=61 time=1.14 ms
```

Now `quagga` should be started at `pathA`:

```
root@pathA:/etc/quagga# /etc/init.d/quagga start
Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfd.
Starting Quagga monitor daemon: watchquagga.
```

Some packets where lost:

```
64 bytes from 10.10.15.2: icmp_req=35 ttl=61 time=0.825 ms
64 bytes from 10.10.15.2: icmp_req=36 ttl=61 time=0.848 ms
64 bytes from 10.10.15.2: icmp_req=37 ttl=61 time=0.832 ms
```

```
64 bytes from 10.10.15.2: icmp_req=38 ttl=61 time=0.833 ms
64 bytes from 10.10.15.2: icmp_req=44 ttl=61 time=0.835 ms
64 bytes from 10.10.15.2: icmp_req=45 ttl=61 time=0.852 ms
64 bytes from 10.10.15.2: icmp_req=46 ttl=61 time=0.921 ms
```

Now router1 shows topology has changed:

```
root@router1:~# ip r
default via 192.168.103.1 dev eth0
10.10.10.0/24 dev eth1  proto kernel  scope link  src 10.10.10.1
10.10.11.0/24 dev eth2  proto kernel  scope link  src 10.10.11.2
10.10.12.0/24 via 10.10.11.1 dev eth2  proto zebra  metric 20
10.10.13.0/24 dev eth3  proto kernel  scope link  src 10.10.13.2
10.10.14.0/24 via 10.10.13.1 dev eth3  proto zebra  metric 20
10.10.15.0/24  proto zebra  metric 30
        nexthop via 10.10.11.1  dev eth2 weight 1
        nexthop via 10.10.13.1  dev eth3 weight 1
192.168.103.0/24 dev eth0  proto kernel  scope link  src 192.168.103.80
```

Now, `quagga` of `pathB` is stopped:

```
64 bytes from 10.10.15.2: icmp_req=160 ttl=61 time=0.936 ms
64 bytes from 10.10.15.2: icmp_req=161 ttl=61 time=1.18 ms
64 bytes from 10.10.15.2: icmp_req=162 ttl=61 time=1.08 ms
64 bytes from 10.10.15.2: icmp_req=163 ttl=61 time=1.12 ms
64 bytes from 10.10.15.2: icmp_req=164 ttl=61 time=1.09 ms
64 bytes from 10.10.15.2: icmp_req=165 ttl=61 time=0.994 ms
64 bytes from 10.10.15.2: icmp_req=166 ttl=61 time=1.12 ms
64 bytes from 10.10.15.2: icmp_req=167 ttl=61 time=1.16 ms
```

Now no packets were lost. Having a look at `router1` routes:

```
root@router1:~# ip r
default via 192.168.103.1 dev eth0
10.10.10.0/24 dev eth1  proto kernel  scope link  src 10.10.10.1
10.10.11.0/24 dev eth2  proto kernel  scope link  src 10.10.11.2
10.10.12.0/24 via 10.10.11.1 dev eth2  proto zebra  metric 20
10.10.13.0/24 dev eth3  proto kernel  scope link  src 10.10.13.2
10.10.14.0/24 via 10.10.11.1 dev eth2  proto zebra  metric 30
10.10.15.0/24 via 10.10.11.1 dev eth2  proto zebra  metric 30
192.168.103.0/24 dev eth0  proto kernel  scope link  src 192.168.103.80
```

Only `pathA` is active as expected.

And finally `pathB` is started again:

```
root@pathB:~# /etc/init.d/quagga start
Loading capability module if not yet done.
```

```
Starting Quagga daemons (prio:10): zebra ospfd.
Starting Quagga monitor daemon: watchquagga.
```

How many packets were lost?

```
64 bytes from 10.10.15.2: icmp_req=343 ttl=61 time=0.834 ms
64 bytes from 10.10.15.2: icmp_req=344 ttl=61 time=0.890 ms
64 bytes from 10.10.15.2: icmp_req=345 ttl=61 time=0.864 ms
64 bytes from 10.10.15.2: icmp_req=346 ttl=61 time=0.893 ms
64 bytes from 10.10.15.2: icmp_req=347 ttl=61 time=0.915 ms
64 bytes from 10.10.15.2: icmp_req=348 ttl=61 time=0.825 ms
64 bytes from 10.10.15.2: icmp_req=349 ttl=61 time=0.839 ms
64 bytes from 10.10.15.2: icmp_req=350 ttl=61 time=0.946 ms
64 bytes from 10.10.15.2: icmp_req=351 ttl=61 time=0.917 ms
64 bytes from 10.10.15.2: icmp_req=352 ttl=61 time=0.948 ms
```

None. `mtr` should be used to see route followed by the packets:

```
root@host1:~# mtr -c 2 --report 10.10.15.2
HOST: host1                       Loss%   Snt   Last   Avg  Best  Wrst
   StDev
  1.|-- 10.10.10.1                 0.0%     2    0.3   0.4   0.3   0.4
     0.1
  2.|-- 10.10.11.1                 0.0%     2    0.6   0.5   0.5   0.6
     0.1
  3.|-- 10.10.14.2                 0.0%     2    1.0   0.8   0.7   1.0
     0.2
  4.|-- 10.10.15.2                 0.0%     2    0.9   0.8   0.8   0.9
     0.0
```

This sequence is:

1. `router1`
2. `pathA`
3. `router2`
4. `host2`

That's why `pathB` modification doesn't affect now.

It's worth mentioning that equal multi path routing remembers the path for single destination IP addresses. So `router1` has cached that path to 10.10.15.2 is through `pathA` router. Next access using multi path may use `pathB` router and when a timeout occurs, with no access to destination IP, this register is erased from cache. So a trip from `host1` to `host2` will normally use a single path. Coming back packets can use the same or the other path. When using this routers by a lot of hosts, traffic is balanced between both paths.

Chapter 4

Operating System clustering Software

4.1 Ways to provide high available running systems

In this chapter we will cover the possible ways to have a complete Operating System (or host) running not attached to a single hardware machine. Sometimes this is called Infrastructure as a Service (IaaS[44]) because we can think about the infrastructure (the Operating System) to be always up and running and take care only of applications installed there.

Normally this means a way to have virtual machines that can be run on more than one node in a cluster. This virtual machine is only active in one single node but, it this node fails, another one boots the virtual machine.

We said *normally* because it is not needed to have virtual machines for this to work, but it is easier. It is possible to have sleeping hardware hosts waiting for another one to fail and in that case be booted as the failed host. This is very slow an not used normally, as it's also a bit difficult to maintain, but it is still a possible solution to choose among.

For example, openQRM[14] cloud software permits to define hosts and its root partition to be on NFS[51] or ISCSI server. When a new node is added to the cloud it boots by PXE[54] and waits for openQRM to tell it what to boot. openQRM can have this node as spare, and in case another one fails, it becomes active. No need to virtualize anything.

One can also build a system that, in case of a node failure, assigns by PXE the failed system to a spare hardware node, and then triggers a *WakeOnLan*[60] event to boot it.

Anyway, for simplicity and quick restoration, normally cloud software using virtualization is used.

Commercially the most used clustering and virtualization software is:

- VMware VSphere (57% of market share[42])
- Microsoft Hyper-V (28% of market share[42])
- Citrix XenServer
- RedHat Enterprise Virtualization

All these systems have:

- A pool of hardware nodes with virtualization capabilities
- A collection of virtual machines to be running
- A management system to add, monitor, start and stop those virtual machines
- Capability to start virtual machines in other nodes in case of a node failure
- Possibility to live migrate virtual machines between nodes, may be automatically to keep all nodes the same busy.

We will cover non commercial GNU/Linux options in next section, but it is worth mentioning Openstack. It's an open source cloud software solution initially sponsored by NASA and Rackspace an now is supported by companies like AT&T, HP, IBM, Dell, Cisco, Intel, Juniper, Ericsson, Paypal... and about 300 more companies including VMWare, Citrix and RedHat which possess their own cloud system. This great support seems to mean a great future.

4.2 The GNU/Linux way

In GNU/Linux we can use an out-of-the-box IaaS solution, or we can build our own. In next sections we will cover:

- virtualization methods
- software useful to build cluster of nodes providing those virtual machines as services
- and complete IaaS solutions.

4.2.1 Virtualization

There are three types of virtualization methods:

- Full virtualization.
- Para-virtualization.
- Containers.

4.2.1.1 Full virtualization

A complete computer is emulated, Operating system running there doesn't need to be modified.

Any Operating system can be run using this virtualization method (Windows, UNIX, ...), it even may not know it is running as a virtual machine.

It's the most resource consumption method as it requires emulating the whole system, but also the easiest to make all applications work.

Some examples are:

- VMware (commercial but free personal usage)
- Virtualbox
- KVM/Qemu

4.2.1.2 Para virtualization

For operating systems that can be modified (normally only open sourced ones), the O.S. is adapted to run as a virtual machine, so is less resource eager than a full virtualized system.

Also, some components can be modified in real time, like CPU or memory size.

One example is Xen.

4.2.1.3 Containers

It is not really virtualization, sometimes they are called jails in other Operating Systems (Solaris or BSD derivatives). All files needed for the system live in a folder and the kernel opens a new instance of the whole Operating System using those files.

It is isolated from other containers and the host node itself. It has its own network devices, RAM size and CPU limits, but no new kernel is booted.

It's the better choice for mass provision machines as it is the least resource consumer virtualization method.

Some examples are:

- OpenVZ (or commercially Virtuozzo containers)
- LXC (Linux Containers)
- UML (User Mode Linux)

4.2.2 Cluster applications

For clustering applications we mean the software needed for a host to know if other hosts are there and what are they doing. A group or cluster of nodes are supposed to provide services. Nodes talk between each other to know if they are alive and providing services. In case of a node failure, the other nodes take over service supply. Each node sends heartbeats to foreign nodes to notify it is alive. When heartbeats don't arrive, the other nodes suppose this node is dead, to assure this they also try to isolate (fence) this node by powering it down with the help of any method:

- its motherboard BIOS (IPMI)
- the switch it is connected to
- the UPS powering it up

Also there are techniques to assure there are not *two partitions* of the cluster giving service. In case of network failure, it is possible that two partitions exist, and some nodes only see the ones on the same partition. This can provide a problem of data corruption, so usually clusters are only operable in case that more that half of the nodes are present, this way it's impossible to have two running partitions. The bigger partition is said to have quorum. Only if quorum is present, the service is given.

There are many applications in GNU/Linux that can provide cluster services (also named resources). May be simpler one is ancient Heartbeat[23]. For resource management an application named Pacemaker[7] is used.

Corosync[8], also known as OpenAIS, is another alternative.

RedHat has put together some software in RedHat Cluster Suite[38] (Corosync, rgmanager, lucci, ricci) to provide a complete system that keeps track of nodes availability and defines services (resources), checks for availability, provides service migration, etc.

4.2.3 Out-of-the-box IaaS solutions

Everybody talks about the Cloud, and today a lot of companies are trying to provide complete IaaS solutions. In the open source arena there are also a lot of opponents and everything is evolving quickly.

This solutions support some of the virtualization methods (normally only full virtualization, KVM is the preferred method for the majority)

Some alternatives are:

- OpenStack[29]
- Cloudstack[2]
- Eucalyptus[15]
- OpenNebula[28]
- OpenQRM[14]

4.3 Practical Example

4.3.1 Description

A cluster of three nodes will be built using RHCS (cman/rgmanager) software. There will be a single service that will be an Openvz virtual machine.

Documentation will begin having:

- One ISCSI server that will provide an ISCSI target that will be virtual machine root file system. This installation will not be covered as it is already done in section 2.4.
- Three nodes that will be forming the cluster, as RHCS software will be used, a RedHat related distribution will be installed, vanilla CentOS will be running on them.

- All nodes will have bridges enabled on network interfaces, to be able to add virtual machines to those bridges. The bridge will be `br0` and will have `eth0` as first slave. Virtual machines will have its network devices also as slaves of `br0`.
- Firewall and Selinux is disabled on all nodes.
- All hosts will have `/etc/hosts` configured so they know each other. This is the content:

```
192.168.103.97          iscsi-server
192.168.103.85          node0
192.168.103.98          node1
192.168.103.92          node2
```

It is needed to:

- Change kernel on three nodes to support OpenVZ containers
- Install and build a cluster on the three nodes.
- Define and install our virtual machine
- Associate this virtual machine to the cluster to provide High Availability

For testing, the node running our virtual machine will be shut down, and it will be seen how fast it restarts in another node. Downtime will be measured to evaluate MTTR.

In figure 4.1 a diagram of this example can be seen.

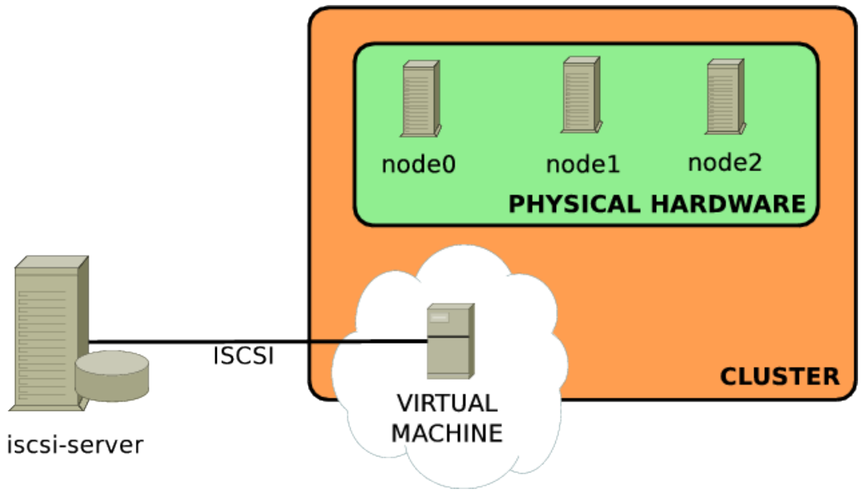


Figure 4.1: Three-node cluster

4.3.2 Installation

4.3.2.1 Changing kernel to support OpenVZ containers

Following official OpenVZ[30] documentation, it can be installed.

First OpenVZ yum repository is added:

```
# wget -P /etc/yum.repos.d/ http://ftp.openvz.org/openvz.repo
--2013-12-02 12:18:44--  http://ftp.openvz.org/openvz.repo
Resolviendo ftp.openvz.org... 199.115.104.11, 2620:e6::104:11
Connecting to ftp.openvz.org|199.115.104.11|:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 2120 (2,1K) [text/plain]
Saving to: '/etc/yum.repos.d/openvz.repo'

100%[=====================================>] 2.120       --.-K/s   in 0s

2013-12-02 12:18:44 (187 MB/s) - '/etc/yum.repos.d/openvz.repo' saved
   [2120/2120]
```

Then repo key is added to the system, so packages are trusted:

```
# rpm --import http://ftp.openvz.org/RPM-GPG-Key-OpenVZ
```

New kernel is installed:

```
# yum install vzkernel
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.udl.es
 * extras: ftp.udl.es
 * openvz-kernel-rhel6: openvz.proserve.nl
 * openvz-utils: openvz.proserve.nl
 * updates: ftp.udl.es
openvz-kernel-rhel6                                    |  951 B     00:00
openvz-kernel-rhel6/primary                            | 3.4 kB     00:00
openvz-kernel-rhel6                                                 12/12
openvz-utils                                           |  951 B     00:00
openvz-utils/primary                                   | 9.5 kB     00:00
openvz-utils                                                        38/38
Setting up Install Process
Resolving Dependencies

[...]

Installed:
  vzkernel.x86_64 0:2.6.32-042stab083.2
Dependency Updated:
  kernel-firmware.noarch 0:2.6.32-431.el6
Complete!
```

And `vzctl` utility:

```
# yum install vzctl
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.udl.es
 * extras: ftp.udl.es
 * openvz-kernel-rhel6: openvz.just-hosting.ru
 * openvz-utils: openvz.just-hosting.ru
 * updates: ftp.udl.es
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package vzctl.x86_64 0:4.6.1-1 will be installed

[...]

Installed:
  vzctl.x86_64 0:4.6.1-1

Dependency Installed:

[ ... ]

Dependency Updated:
  db4.x86_64 0:4.7.25-18.el6_4   db4-utils.x86_64 0:4.7.25-18.el6_4
  glibc.x86_64 0:2.12-1.132.el6  glibc-common.x86_64 0:2.12-1.132.el6

Complete!
```

Then the system is rebooted, kernel is checked to have changed and `openvz` service has to be enabled:

```
# uname -r
2.6.32-042stab083.2
# service vz status
OpenVZ is running...
```

Which is correct.

It is just needed to add a file to be able to connect virtual machines to this network. To be able to add their networks to bridge `br0`, it is needed to create the file `/etc/vz/vznet.conf` on those three nodes with this content:

```
#!/bin/bash
EXTERNAL_SCRIPT="/usr/sbin/vznetaddbr"
```

4.3.2.2 Installing and configuring Cman

`cman` package can be installed:

```
# yum install cman
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.udl.es
 * extras: ftp.udl.es
 * openvz-kernel-rhel6: mirror.softaculous.com
 * openvz-utils: mirror.softaculous.com
 * updates: ftp.udl.es
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package cman.x86_64 0:3.0.12.1-59.el6 will be installed

[...]

Installed:
  cman.x86_64 0:3.0.12.1-59.el6

Dependency Installed:

[ ...]

Complete!
```

Now, `/etc/cluster/cluster.conf` should be configured to define node members. This file will have this content on all three nodes:

```
<?xml version="1.0"?>
<cluster name="uoclab" config_version="1">

<clusternodes>
    <clusternode name="node0" votes="1" nodeid="1">
      <fence> <method name="1"> <device name="nofence"/> </method> </fence>
    </clusternode>
    <clusternode name="node1" votes="1" nodeid="2">
      <fence> <method name="1"> <device name="nofence"/> </method> </fence>
    </clusternode>
    <clusternode name="node2" votes="1" nodeid="3">
      <fence> <method name="1"> <device name="nofence"/> </method> </fence>
    </clusternode>
  </clusternodes>

  <fencedevices>
    <fencedevice agent="nofence" name="nofence"/>
  </fencedevices>
</cluster>
```

Fencing has been disabled in our tests. For *nofence fencing* to work we need to create the file `/usr/sbin/nofence` with this content:

```
#!/bin/sh
exit 0
```

Now `cman` can be started:

```
# service cman start
Starting cluster:
   Checking if cluster has been disabled at boot...        [  OK  ]
   Checking Network Manager...                             [  OK  ]
   Global setup...                                         [  OK  ]
   Loading kernel modules...                               [  OK  ]
   Mounting configfs...                                    [  OK  ]
   Starting cman...                                        [  OK  ]
   Waiting for quorum...                                   [  OK  ]
   Starting fenced...                                      [  OK  ]
   Starting dlm_controld...                                [  OK  ]
   Tuning DLM kernel config...                             [  OK  ]
   Starting gfs_controld...                                [  OK  ]
   Unfencing self...
```

Let's tell the system to start `cman` when it starts:

```
# chkconfig cman on
```

If `cman_tool` utilities are used on any node, all three nodes and cluster status can be seen connected:

```
[root@node0 cluster]# cman_tool nodes
Node  Sts   Inc   Joined               Name
   1   M     44   2013-12-02 19:08:57  node0
   2   M     48   2013-12-02 19:09:01  node1
   3   M     52   2013-12-02 19:09:07  node2
[root@node0 cluster]# cman_tool status
Version: 6.2.0
Config Version: 1
Cluster Name: uoclab
Cluster Id: 17169
Cluster Member: Yes
Cluster Generation: 52
Membership state: Cluster-Member
```

```
Nodes: 3
Expected votes: 3
Total votes: 3
Node votes: 1
Quorum: 2
Active subsystems: 7
Flags:
Ports Bound: 0
Node name: node0
Node ID: 1
Multicast addresses: 239.192.67.84
Node addresses: 192.168.103.85
```

4.3.2.3 Installing the virtual machine

ISCSI client utilities are needed on all three nodes:

```
# yum install iscsi-initiator-utils
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.udl.es
 * extras: ftp.udl.es
 * openvz-kernel-rhel6: mirrors.ircam.fr
 * openvz-utils: mirrors.ircam.fr
 * updates: ftp.udl.es
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package iscsi-initiator-utils.x86_64 0:6.2.0.873-10.el6 will be
   installed
--> Finished Dependency Resolution

Dependencies Resolved

[...]

Installed:
  iscsi-initiator-utils.x86_64 0:6.2.0.873-10.el6

Complete!
```

Now it is also needed to login to the target ISCSI iqn on all nodes:

```
# iscsiadm --mode discovery --type sendtargets --portal iscsi-server
Iniciando iscsid:                                          [  OK  ]
```

```
192.168.103.97:3260,1 iqn.2013-11.edu.uoc:ha-disk
# iscsiadm --mode node --targetname iqn.2013-11.edu.uoc:ha-disk --portal
   iscsi-server --login
Logging in to [iface: default, target: iqn.2013-11.edu.uoc:ha-disk, portal:
    192.168.103.97,3260] (multiple)
Login to [iface: default, target: iqn.2013-11.edu.uoc:ha-disk, portal:
   192.168.103.97,3260] successful.
```

Disk is `/dev/sda` (root file system is `/dev/vda` in our laboratory environments). Disk can now be formatted, just from one node:

```
[root@node0 ~]# mkfs.ext4 -F /dev/sda
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
160000 inodes, 640000 blocks
32000 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=658505728
20 block groups
32768 blocks per group, 32768 fragments per group
8000 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 29 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

For openvz installation, it is needed an operating system template. A minimal Debian template will be downloaded:

```
[root@node0 ~]# wget http://download.openvz.org/template/precreated/contrib
   /debian-7.0-amd64-minimal.tar.xz -O /vz/template/cache/debian-7.0-amd64-
   minimal.tar.xz
--2013-12-02 19:42:48--  http://download.openvz.org/template/precreated/
   contrib/debian-7.0-amd64-minimal.tar.xz
Resolving download.openvz.org... 199.115.104.11, 2620:e6::104:11
Connecting to download.openvz.org|199.115.104.11|:80... connected.
HTTP request sent, awaiting response... 200 OK
```

```
Length: 42428832 (40M) [application/x-xz]
Saving to: '/vz/template/cache/debian-7.0-amd64-minimal.tar.xz'

100%[========================================>] 42,428,832   442K/s   in
   97s

2013-12-02 19:44:26 (425 KB/s) - '/vz/template/cache/debian-7.0-amd64-
   minimal.tar.xz' saved [42428832/42428832]
```

A virtual machine can now be created using this template with an example id of *100* and host name *virtualmachine*:

```
[root@node0 ~]# vzctl create 100 --ostemplate debian-7.0-amd64-minimal --
   hostname virtualmachine
Creating container private area (debian-7.0-amd64-minimal)
Performing postcreate actions
CT configuration saved to /etc/vz/conf/100.conf
Container private area was created
```

Its network interface can be attached to host `br0`:

```
[root@node0 ~]# vzctl set 100 --netif_add eth0,,,,br0 --save
CT configuration saved to /etc/vz/conf/100.conf
```

Disk quotas can be disabled, as this virtual machine will has its own partition, and it is not needed to limit more the space to use:

```
[root@node0 ~]# vzctl set 100 --diskquota no --save
CT configuration saved to /etc/vz/conf/100.conf
```

It can be started:

```
[root@node0 ~]# vzctl start 100
Starting container...
Container is mounted
Setting CPU units: 1000
Configure veth devices: veth100.0
Adding interface veth100.0 to bridge br0 on CT0 for CT100
Container start in progress...
```

This host can now be entered:

```
[root@node0 ~]# vzctl enter 100
entered into CT 100
root@virtualmachine:/#
```

`etc/network/interfaces` should be edited to add `eth0` configuration:

```
auto eth0
iface eth0 inet dhcp
```

Network can be started, a ping to an Internet host can be sent and its console can be left:

```
root@virtualmachine:/# ifup eth0
Internet Systems Consortium DHCP Client 4.2.2
Copyright 2004-2011 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/00:18:51:28:62:56
Sending on   LPF/eth0/00:18:51:28:62:56
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 7
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPOFFER from 192.168.103.1
DHCPACK from 192.168.103.1
bound to 192.168.103.99 -- renewal in 245 seconds.
root@virtualmachine:/# ping -c 1 www.google.com
PING www.google.com (173.194.34.244) 56(84) bytes of data.
64 bytes from mad01s09-in-f20.1e100.net (173.194.34.244): icmp_req=1 ttl=56
    time=32.5 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 32.515/32.515/32.515/0.000 ms
root@virtualmachine:/# exit
logout
exited from CT 100
[root@node0 ~]#
```

The virtual machine has its files in `/vz/private/100` in `node0` file system. The virtual machine should be stopoed now and its data can be moved to ISCSI partition:

```
[root@node0 ~]# vzctl stop 100
Stopping container ...
Container was stopped
Container is unmounted
[root@node0 ~]# mount /dev/sda /mnt/
[root@node0 ~]# mv /vz/private/100/* /mnt/
```

Now, to tell OpenVZ that this partition should me mounted, `/etc/vz/conf/100.mount` is created with this content:

```
#!/bin/sh
```

```
mount /dev/sda /vz/root/100
```

And `/etc/vz/conf/100.umount` with this content:

```
#!/bin/sh
umount /vz/root/100
```

The virtual machine will be started again to check if the file system mounted is `/dev/sda` and virtual machine has network connectivity:

```
[root@node0 ~]# vzctl start 100
Starting container...
Container is mounted
Setting CPU units: 1000
Configure veth devices: veth100.0
Adding interface veth100.0 to bridge br0 on CT0 for CT100
Container start in progress...
[root@node0 ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup-lv_root
                       2620520   1030964   1456436  42% /
tmpfs                   120088     16416    103672  14% /dev/shm
/dev/vda1               495844     54116    416128  12% /boot
/dev/sda               2519792    299856   2091936  13% /mnt
/dev/sda               2519792    299856   2091936  13% /vz/root/100
[root@node0 ~]# vzctl enter 100
entered into CT 100
root@virtualmachine:/# ping -c 1 www.google.com
PING www.google.com (173.194.34.241) 56(84) bytes of data.
64 bytes from mad01s09-in-f17.1e100.net (173.194.34.241): icmp_req=1 ttl=56
    time=54.4 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 54.487/54.487/54.487/0.000 ms
root@virtualmachine:/# exit
logout
exited from CT 100
[root@node0 ~]# vzctl stop 100
Stopping container ...
Container was stopped
Container is unmounted
```

Now configuration is replicated in the other two nodes. It is needed to create:

- `/vz/private/100`

- /vz/root/100

And those files should be copied:

- /etc/vz/100.conf
- /etc/vz/100.mount
- /etc/vz/100.umount

And now, this virtual machine can be started on any node, but only one at once. So RHCS should be responsible of starting this service.

4.3.2.4 Installing `rgmanager` and defining our cluster service for High Availability

First `rgmanager` is installed on all three nodes:

```
# yum install rgmanager
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.udl.es
 * extras: ftp.udl.es
 * openvz-kernel-rhel6: mirrors.ircam.fr
 * openvz-utils: mirrors.ircam.fr
 * updates: ftp.udl.es
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package rgmanager.x86_64 0:3.0.12.1-19.el6 will be installed

[ ... ]

Installed:
  rgmanager.x86_64 0:3.0.12.1-19.el6

Dependency Installed:

[ ... ]

Complete!
```

Now, the system is told to start `rgmanager` when it starts:

```
# chkconfig rgmanager on
```

`rgmanager` comes with some resource agents in `/usr/share/cluster` folder.

Those scripts are responsible of starting, stopping, migrating and checking status of each service they are managing. An specific resource agent will be added for openvz container management. For doing so, it will be downloaded and modified the one used by LBVM[12] project.

The content of `/usr/share/cluster/openvz.sh` will then be:

```
#!/bin/bash

#
# Script to handle an OpenVZ Virtual Machine
#

LC_ALL=C
LANG=C
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export LC_ALL LANG PATH

meta_data()
{
    cat <<EOT
<?xml version="1.0"?>
<resource-agent version="rgmanager 2.0" name="openvz">
    <version>1.0</version>

    <longdesc lang="en">
        Defines an OpenVZ virtual environment
    </longdesc>
    <shortdesc lang="en">
        Defines an OpenVZ virtual environment
    </shortdesc>

    <parameters>
        <parameter name="name" unique="1" primary="1">
            <longdesc lang="en">
                Name (VEID)
            </longdesc>
            <shortdesc lang="en">
                Name (VEID)
            </shortdesc>
            <content type="string"/>
        </parameter>

        <parameter name="domain">
            <longdesc lang="en">
                Fail over domains define lists of cluster members
                to try in the event that the host of the virtual machine
                fails.
            </longdesc>
            <shortdesc lang="en">
                Cluster Fail Over Domain
            </shortdesc>
            <content type="string"/>
        </parameter>

        <parameter name="autostart">
            <longdesc lang="en">
                If set to yes, this resource group will automatically be
                   started
                after the cluster forms a quorum.  If set to no, this
                   virtual
                machine will start in the 'disabled' state after the
                   cluster
                forms a quorum.
            </longdesc>
            <shortdesc lang="en">
                Automatic start after quorum formation
            </shortdesc>
            <content type="boolean"/>
        </parameter>
    </parameters>

    <actions>
        <action name="start" timeout="30"/>
        <action name="stop" timeout="120"/>

        <action name="status" interval="30s" timeout="10"/>
        <action name="monitor" interval="30s" timeout="10"/>

        <action name="meta-data" timeout="0"/>
        <action name="verify-all" timeout="0"/>
    </actions>
</resource-agent>
EOT
}


build_vz_cmdline_nono()
{
        #
        # Virtual domains should never restart themselves when
        # controlled externally; the external monitoring app
        # should.
        #
        declare cmdline="restart=\"never\""
        declare varp val temp

        #
        # Transliterate the OCF_RESKEY_* to something the xm
        # command can recognize.
        #
        for var in ${!OCF_RESKEY_*}; do
                varp=${var/OCF_RESKEY_/}
                val=`eval "echo \\$$var"`

                case $varp in
                recovery|autostart|domain)
                        ;;
                name)   # Do nothing with name; add it later
                        ;;
                *)
                        cmdline="$cmdline $varp=\"$val\""
                        ;;
                esac
        done

        if [ -n "$OCF_RESKEY_name" ]; then
                cmdline="$OCF_RESKEY_name $cmdline"
        fi

        echo $cmdline
}



#
# Start a virtual machine given the parameters from
# the environment.
#
start()
{
        if [ ! -f /etc/vz/conf/$OCF_RESKEY_name.conf ]; then
                return 6; # No configuration file found
        fi

        eval vzctl start $OCF_RESKEY_name
        return $?
```

```
}

#
# Stop a VM.
#
stop()
{

        eval vzctl stop $OCF_RESKEY_name
        return $?

}

#
# Simple status check: Find the VM in the list of running
# VMs
#
status()
{
        #vzctl status $OCF_RESKEY_name |awk '{print $5}' &> /dev/null

        if [ `vzctl status $OCF_RESKEY_name |awk {'printf $5'}` == "running
           " ]; then
                return 0;
        fi

        return 7; # OCF_NOT_RUNNING
}

case $1 in
        start)
                start
                exit $?
                ;;
        stop)
                stop #shutdown destroy
                exit $?
                ;;
        status|monitor)
                status
                exit $?
                ;;
        meta-data)
                meta_data
                exit 0
                ;;
        *)
                echo "usage: $0 {start|stop|status|monitor|meta-data|verify
                   -all}"
                exit 1
                ;;
esac
```

Finally, it is needed to modify `/etc/cluster/cluster.conf` to add this virtual machine as a cluster service.

Those modifications should be done:

- Increase config_version number
- Add the resource in `rm` section

This file is modified and left it with this content:

```
<?xml version="1.0"?>
<cluster name="uoclab" config_version="2">

<clusternodes>
    <clusternode name="node0" votes="1" nodeid="1">
      <fence> <method name="1"> <device name="no fence"/> </method> </fence
          >
    </clusternode>
    <clusternode name="node1" votes="1" nodeid="2">
      <fence> <method name="1"> <device name="no fence"/> </method> </fence
          >
    </clusternode>
    <clusternode name="node2" votes="1" nodeid="3">
      <fence> <method name="1"> <device name="no fence"/> </method> </fence
          >
    </clusternode>
  </clusternodes>

  <fencedevices>
    <fencedevice agent="no fence" name="no fence"/>
  </fencedevices>
  <rm>
    <resources/>
    <service name="virtualmachine">
      <openvz name="100"/>
    </service>
  </rm>
</cluster>
```

After copying the file to all three nodes it should be executed in one of them the command to reload configuration on all nodes:

```
# cman_tool -S -r version
```

This command can be issued to check status in one node:

```
[root@node0 ~]# cman_tool status
Version: 6.2.0
Config Version: 2
Cluster Name: uoclab
Cluster Id: 17169
Cluster Member: Yes
Cluster Generation: 152
Membership state: Cluster-Member
Nodes: 3
Expected votes: 3
Total votes: 3
Node votes: 1
Quorum: 2
Active subsystems: 7
Flags:
Ports Bound: 0
Node name: node0
Node ID: 1
Multicast addresses: 239.192.67.84
Node addresses: 192.168.103.85
```

OK, version 2 is loaded. `rgmanager` comes with some utilities to check resources. This command can be issued to see service status:

```
[root@node0 cluster]# clustat
Cluster Status for uoclab @ Tue Dec  3 10:39:08 2013
Member Status: Quorate

 Member Name                             ID   Status
 ------ ----                            ---- ------
 node0                                      1 Online, Local, rgmanager
 node1                                      2 Online, rgmanager
 node2                                      3 Online, rgmanager

 Service Name                  Owner (Last)                  State
 ------- ----                  ----- ------                  -----
 service:virtualmachine        node2                         started
```

Service has been started on `node2`. To check it:

```
[root@node2 cluster]# vzlist
      CTID      NPROC STATUS    IP_ADDR         HOSTNAME
       100         10 running   -               virtualmachine
[root@node2 cluster]# vzctl enter 100
entered into CT 100
root@virtualmachine:/# ping www.google.com
PING www.google.com (173.194.41.244) 56(84) bytes of data.
64 bytes from mad01s15-in-f20.1e100.net (173.194.41.244): icmp_req=1 ttl=56
    time=30.2 ms
^C
--- www.google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 30.253/30.253/30.253/0.000 ms
```

4.3.3 Failure test

In this test `node2` will be suddenly powered down (the one that is running this virtual machine). It will seen what happens and downtime will be evaluated.

For this evaluation `iscsi-server` will ping to our virtual machine IP address, which is 192.168.103.99.

While pinging, `node2` will be hang up and the seconds of downtime will be checked:

```
64 bytes from 192.168.103.99: icmp_req=34 ttl=64 time=0.815 ms
64 bytes from 192.168.103.99: icmp_req=35 ttl=64 time=0.505 ms
64 bytes from 192.168.103.99: icmp_req=36 ttl=64 time=0.446 ms
64 bytes from 192.168.103.99: icmp_req=37 ttl=64 time=0.435 ms
64 bytes from 192.168.103.99: icmp_req=38 ttl=64 time=0.387 ms
64 bytes from 192.168.103.99: icmp_req=39 ttl=64 time=0.506 ms
64 bytes from 192.168.103.99: icmp_req=69 ttl=64 time=0.953 ms
64 bytes from 192.168.103.99: icmp_req=70 ttl=64 time=0.377 ms
64 bytes from 192.168.103.99: icmp_req=71 ttl=64 time=0.336 ms
```

There have been 30 seconds of downtime between:

- detecting `node2` failure
- starting again the service in another node
- and pure virtual machine start up.

`clustat` shows now service is running on `node0`:

```
[root@node1 ~]# clustat
Cluster Status for uoclab @ Tue Dec  3 11:03:11 2013
Member Status: Quorate
```

```
 Member Name                              ID   Status
 ------ ----                            ---- ------
 node0                                      1 Online, rgmanager
 node1                                      2 Online, Local, rgmanager
 node2                                      3 Offline

 Service Name                     Owner (Last)                    State
 ------- ----                     ----- ------                    -----
 service:virtualmachine           node0
    started
```

So, having a look at `node0` cluster logs:

```
Dec  3 11:00:25 node0 corosync[8202]:   [QUORUM] Members[2]: 1 2
Dec  3 11:00:25 node0 corosync[8202]:   [TOTEM ] A processor joined or left
    the membership and a new membership was formed.
Dec  3 11:00:25 node0 kernel: [ 5951.346757] dlm: closing connection to
   node 3
Dec  3 11:00:25 node0 rgmanager[8445]: State change: node2 DOWN
Dec  3 11:00:25 node0 corosync[8202]:   [CPG   ] chosen downlist: sender r
   (0) ip(192.168.103.85) ; members(old:3 left:1)
Dec  3 11:00:25 node0 corosync[8202]:   [MAIN  ] Completed service
   synchronization, ready to provide service.
Dec  3 11:00:25 node0 fenced[8257]: fencing node node2
Dec  3 11:00:25 node0 fenced[8257]: fence node2 success
Dec  3 11:00:26 node0 rgmanager[8445]: Taking over service service:
   virtualmachine from down member node2
Dec  3 11:00:26 node0 kernel: [ 5952.911610] EXT4-fs (sda): recovery
   complete
Dec  3 11:00:26 node0 kernel: [ 5952.912405] EXT4-fs (sda): mounted
   filesystem with ordered data mode. Opts:
Dec  3 11:00:26 node0 kernel: [ 5952.923359] CT: 100: started
Dec  3 11:00:26 node0 kernel: [ 5953.311713] device veth100.0 entered
   promiscuous mode
Dec  3 11:00:26 node0 kernel: [ 5953.313998] veth100.0 adding interface
   with same address as a received packet
Dec  3 11:00:26 node0 kernel: [ 5953.314108] br0: port 2(veth100.0)
   entering forwarding state
Dec  3 11:00:27 node0 rgmanager[8445]: Service service:virtualmachine
   started
```

It can be seen how it detects `node2` failure, mounts partition and starts virtual machine.

4.3.4 Possible improvements

RPO can be of zero seconds, but RTO will be bigger than that. Downtime can be divided in those sections:

- node failure detection. Heartbeats can be fine tuned so downtime can be as low as 1-2 seconds. So this time is negligible.
- service startup. In this example in the same second that was detected, launch was triggered. So this time is even more negligible.
- Real service start up. This is the longer period to deal of. Bridge configuration wasn't modified on nodes, STP can be disabled and join time reduced. Normally there is a time to check if there exists a network loop. Furthermore, just the network was configured, depending on the services that are to be launch in real cases (may be application servers), there will be longer downtime.

Chapter 5

Application Level High Availability Techniques

5.1 Overview

So far, there has been a presentation of available solutions that need no application modification to be highly available. Those are general solutions for every application, but might not be well suited for organizations who want to provide a service by means of geographically distant servers.

When applications implement High Availability, it is possible to achieve no service disruption (MTTR equal to zero) because a server failure can be hidden by the application that silently connects to another server and service seems to be always on. This techniques are smart and inexpensive in hardware, but they are sometimes difficult to implement, and every solution can not be shared with other applications or services.

Any application has to implement its own system and it is really tied to how the service works, so it is difficult to explain here a way to achieve this. This will be clear if we look at some examples in next section.

5.2 Example applications

5.2.1 ISC dhcpd

ISC has developed a dhcp server widely installed in Linux systems. This application provides High Availability[20]. One can set two servers up, one that acts as primary and other one as secondary that talk each other.

Dhcp service works by giving IP addresses to hosts that ask for them. The application can know if it running as primary server, so it replies to DHCP requests, or can know if it is secondary, waiting to give service in case primary stops working. If that is the case, they will begin answering to DHCP requests and will know previous given addresses to hosts. Hosts asking for IP addresses will not know a new server is replying now.

5.2.2 SMTP fail over

SMTP protocol is designed to allow High Availability by defining MX records[50] in DNS service. To configure which servers will receive email, domain names have to configure those records in DNS. One can use to records to point to two different servers and optionally different preferences of delivery.

SMTP clients (Mail Transport Units) in order to deliver an email to an specific domain, will check for those MX records. They will first try to connect to one SMTP server (normally preferred one) and if that connection fails they will connect to next SMTP server. Email users will not notice that and service is always up.

5.2.3 DNS service

DNS[47] is another example. Normally hosts have more than one DNS server configured in their system. If first DNS server timeouts to respond to DNS queries, the client (normally the Operating System) asks another DNS server. Applications that ask for an IP address, like Web browsers for example, will not know if first or secondary DNS server is used, and service is also always on.

5.2.4 Mysql servers

Mysql servers can make use of replication[32]. A secondary server can receive all queries that are sent to primary server. Both servers make the same modifications to their data. In case the primary server fails, the mysql clients can make use of secondary server.

For applications to reconnect to new servers there are different approaches that can be followed, the should detect a connection failure and the should try to reconnect. There are some ways to tell them to use a new Mysql server:

- applications can have a list of servers to connect, in case of failure can follow their list to connect to another one
- application can ask a DNS server for the actual IP to connect. DNS server can be configured to detect server failure and reply with new IP address. DNS Time To Live (TTL) should be low to quickly propagate changes.
- they can connect to the same IP address, and software like Heartbeat[23] can be used to add the same Virtual IP address to a new mysql server. This will only be possible in case both servers are on the same subnet.

And may be someone can think of a fourth way to achieve reconnection. There are no limited ways if redundancy is configured in software.

5.3 Usual techniques

5.3.1 Client/server reconnection

Clients can have a list of servers to connect, and in case some of them fails, use another one in the list. Applications know this reconnection is necessary:

- when new connections are made. May be the service makes a new TCP connection for every application event, every time can be consulted this list to know where to connect to.
- When a TCP connection is lost. In a open TCP connection, keep alive packets can be sent. In case some of them do not arrive back, the client can reconnect to the same or other server in the list.
- when an UDP packet response times out. Application can send UDP packets and wait for responses. A timeout can be configured and, in case this timeout is reached, a new server in the list can be contacted. This is used for example by DNS system.

This list or pool of servers can be hard coded in the application or a DNS SRV[58] can be used. By DNS a pool of servers and ports can be defined. The clients can check those records to receive the list of servers that provide a server. This will be used in our practical example in this chapter, section 5.5.

5.3.2 Farms of servers with no clustered applications

A pool of numerous servers can be used to provide a service[56]. This normally is used to increase performance rather than redundancy, but also provides redundancy.

Normally a proxy talks to clients that ask for a service. This proxy sometimes is called a head node. To provide high availability this node should be replicated and should give service in case primary server fails. The easiest way is to use an application like Heartbeat[23] to share a Virtual IP address between both head nodes, but any high available IaaS technique can be used to provide the head node.

The head node will then make use of the farm nodes to provide service, balancing the load among those servers. Then each client is talking to a particular node in the farm, queries are balanced through all nodes.

Some examples of this are:

- Services balanced by Linux Virtual Server[24] or balance[19]. At TCP level. The head node balances every new TCP connection to a new server. The head node itself can detect a node failure and send a TCP connection to only active nodes.
- Application server farms. The head node is a web server, for example Apache, that balances HTTP requests to application server nodes. There are many languages to chose[46]:
  - Java: Weblogic from Oracle, WebSphere from IBM, Jboss from Redhat...
  - Python: Zope, Paste...
  - PHP: Zend

5.3.3 Applications developed on top of cluster frameworks

Some other farms of servers provide a framework or API to build applications on top of it. Every instruction can be balanced to different nodes, even parallel executions can be done on different servers to provide a single calculation. This way high performance applications can be used. In previous section we see a way to balance clients to different nodes. In this section a single client can exist that uses the whole farm of servers. Very powerful applications can be built this way. And the system is highly scalable as more compute nodes can be added to provide more power.

Some examples are:

- Apache's Hadoop[4] for distributed computing. The project literally says:

  Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

- Apache's Cassandra[3] for databases. All data can be partitioned and shared between nodes. All data is replicated between more than one one in the same partition. So single nodes can fail and a good performance and storage level is achieved summing up partitions.

5.4 The GNU/Linux way

In this chapter we will not see a GNU/Linux way as all is implemented at application layer, we have seen GNU/Linux examples in previous chapter and we will see a practical example in next one.

5.5 Practical Example

5.5.1 Description

It will be built an High Available SIP registrar service provided by two Asterisk servers. Two SIP clients will be connected to the service, by DNS SRV they will know the list of possible servers and will register at one.

Both clients will be able to call each other. In case of a server failure, they will keep reachable through the remaining server.

At figure 5.1 can be seen a diagram of the configuration.

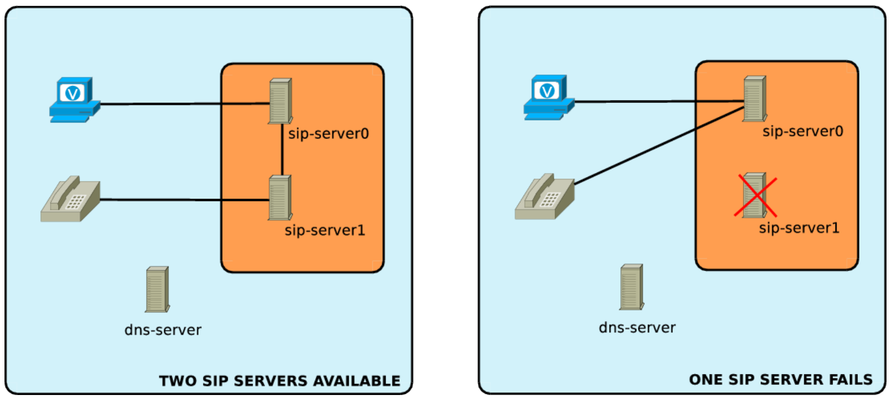


Figure 5.1: SIP service elements

Asterisk configuration is a bit beyond the scope of this document. Only installation and all files configurations will be covered, but further reading will be necessary to understand all concepts.

5.5.2 Installation

It is needed to install:

- two SIP servers, they will be named `sip-registrar0` and `sip-registrar1` and will provide the Service
- to SIP clients, a software and a hardware one that will be calling each other
- a DNS server, named `dns-server`, that will answer with the pool of servers that provide the Service

All hosts will have:

- Debian Operating System installed
- Network configured
- DNS name servers set to ask dns-server with a default search domain of *lab.uoc*.

5.5.2.1 DNS server

First bind name server will be installed:

```
root@dns-server:~# apt-get install bind9
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bind9utils geoip-database libbind9-80 libcap2 libclass-isa-perl libdns88
  libgeoip1 libisc84 libisccc80 libisccfg82 liblwres80 libswitch-perl
  libxml2 perl perl-modules sgml-base xml-core
Suggested packages:
  dnsutils bind9-doc resolvconf ufw geoip-bin perl-doc
  libterm-readline-gnu-perl libterm-readline-perl-perl make
  libpod-plainer-perl sgml-base-doc debhelper
The following NEW packages will be installed:
  bind9 bind9utils geoip-database libbind9-80 libcap2 libclass-isa-perl
  libdns88 libgeoip1 libisc84 libisccc80 libisccfg82 liblwres80
  libswitch-perl libxml2 perl perl-modules sgml-base xml-core
0 upgraded, 18 newly installed, 0 to remove and 0 not upgraded.
Need to get 12.1 MB of archives.
After this operation, 42.8 MB of additional disk space will be used.
Do you want to continue [Y/n]?
Get:1 http://ftp.cica.es/debian/ wheezy/main libcap2 amd64 1:2.22-1.2 [13.6
    kB]

[ ... ]
```

```
Setting up bind9 (1:9.8.4.dfsg.P1-6+nmu2+deb7u1) ...
Adding group ‘bind’ (GID 104) ...
Done.
Adding system user ‘bind’ (UID 102) ...
Adding new user ‘bind’ (UID 102) with group ‘bind’ ...
Not creating home directory ‘/var/cache/bind’.
wrote key file "/etc/bind/rndc.key"
#
[ ok ] Starting domain name service...: bind9.
Setting up perl-modules (5.14.2-21+deb7u1) ...
Setting up libswitch-perl (2.16-2) ...
Setting up perl (5.14.2-21+deb7u1) ...
update-alternatives: using /usr/bin/prename to provide /usr/bin/rename (
   rename) in auto mode
Setting up sgml-base (1.26+nmu4) ...
Setting up xml-core (0.13+nmu2) ...
Processing triggers for sgml-base ...
```

Then, bind needs to be configured to answer to DNS SRV requests replying with those two previous servers. It will be used SIP protocol over UDP transport, so it is needed the DNS record _sip._udp.lab.uoc.

`/etc/bind/named.conf.local` has to be edited, leaving this content:

```
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "lab.uoc" {
        type master;
        file "/etc/bind/db.lab.uoc";
};
```

Then lab.uoc zone has to be edited, the file is `/etc/db.lab.uoc`, as referenced in previous file. This will be its content:

```
;
; BIND data file for lab.uoc zone
;
$TTL    604800
@               IN      SOA     lab.uoc. hostmaster.lab.uoc. (
                                     1         ; Serial
                                604800         ; Refresh
```

```
                                  86400         ; Retry
                                2419200         ; Expire
                                 604800 )       ; Negative Cache TTL
;
@               IN      NS      dns-server

; Hosts
dns-server      IN      A       192.168.103.98
sip-server0     IN      A       192.168.103.101
sip-server1     IN      A       192.168.103.102

; Services
_sip._udp       IN      SRV     0       10      5060    sip-server0
_sip._udp       IN      SRV     0       10      5060    sip-server1
```

The service will be restarted, and responses to DNS queries will be checked:

```
root@dns-server:~# /etc/init.d/bind9 restart
[....] Stopping domain name service...: bind9waiting for pid 3743 to die
. ok
[ ok ] Starting domain name service...: bind9.
root@dns-server:~# host -t srv _sip._udp.lab.uoc 127.0.0.1
Using domain server:
Name: 127.0.0.1
Address: 127.0.0.1#53
Aliases:

_sip._udp.lab.uoc has SRV record 0 10 5060 sip-server1.lab.uoc.
_sip._udp.lab.uoc has SRV record 0 10 5060 sip-server0.lab.uoc.
root@dns-server:~# host -t a sip-server0.lab.uoc 127.0.0.1
Using domain server:
Name: 127.0.0.1
Address: 127.0.0.1#53
Aliases:

sip-server0.lab.uoc has address 192.168.103.101
root@dns-server:~# host -t a sip-server1.lab.uoc 127.0.0.1
Using domain server:
Name: 127.0.0.1
Address: 127.0.0.1#53
Aliases:

sip-server1.lab.uoc has address 192.168.103.102
```

5.5.2.2 SIP servers

First, it is needed to install asterisk software:

```
# apt-get install asterisk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  asterisk-config asterisk-core-sounds-en asterisk-core-sounds-en-gsm
  asterisk-modules asterisk-moh-opsound-gsm asterisk-voicemail autopoint
  binutils build-essential bzip2 ca-certificates cpp cpp-4.7 debhelper
  dpkg-dev fakeroot file freetds-common g++ g++-4.7 gcc gcc-4.7 gettext
  git git-man html2text intltool-debian less libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libasound2 libc-dev-bin
  libc6-dev libcap2 libclass-isa-perl libcroco3 libcurl3 libcurl3-gnutls
  libdpkg-perl liberror-perl libffi5 libfile-fcntllock-perl libflac8
  libgettextpo0 libglib2.0-0 libglib2.0-data libgmime-2.6-0 libgmp10
  libgomp1 libgpgme11 libgsm1 libical0 libiksemel3 libitm1 libjack-jackd2-0
  libjbig0 libjpeg8 libldap-2.4-2 libltdl7 liblua5.1-0 libmagic1
  libmail-sendmail-perl libmpc2 libmpfr4 libneon27-gnutls libodbc1 libogg0
  libopencore-amrnb0 libopencore-amrwb0 libpcre3 libperl5.14 libpng12-0
  libpq5 libpth20 libquadmath0 libradiusclient-ng2 libresample1 librtmp0
  libsaclm3 libsaevt3 libsamplerate0 libsasl2-2 libsasl2-modules
  libsensors4 libsndfile1 libsnmp-base libsnmp15 libsox-fmt-alsa
  libsox-fmt-base libsox2 libspandsp2 libspeex1 libspeexdsp1 libsqlite0
  libsrtp0 libssh2-1 libstdc++6-4.7-dev libswitch-perl libsybdb5
  libsys-hostname-long-perl libtiff4 libtimedate-perl libunistring0
  libvorbis0a libvorbisenc2 libvorbisfile3 libvpb0 libwavpack1 libxml2
  linux-libc-dev make manpages-dev module-assistant openssl patch perl
  perl-modules po-debconf rsync sgml-base shared-mime-info sox
  vpb-driver-source xml-core
Suggested packages:

[...]

Processing triggers for ca-certificates ...
Updating certificates in /etc/ssl/certs... 158 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d....done.
Processing triggers for sgml-base ...
```

Now it is time to add new VoIP users to be able to register. This will be appended to `/etc/asterisk/users.conf` file on both SIP servers:

```
[dundi]
type = friend
```

```
dbsecret = dundi/secret
disallow=all
allow=alaw
context=internal

[100]
type = user
secret = 100
host=dynamic
disallow=all
allow=alaw
context=internal

[101]
type = user
secret = 101
host=dynamic
disallow=all
allow=alaw
context=internal
```

In asterisk, it is possible to keep a *context* with all extensions that are registered. It is needed to edit `/etc/asterisk.sip.conf` and left only with this content:

```
[general]
context=internal
regcontext=sipregistrations
```

Now, DUNDi should be configured between both Asterisk servers. First, it is necessary to generate a secure key to talk between both servers. This command can be issued on just one SIP server:

```
# cd /usr/share/asterisk/keys
# astgenkey -n astlabkey

This script generates an RSA private and public key pair
in PEM format for use by Asterisk.  You will be asked to
enter a passcode for your key multiple times.  Please
enter the same code each time.  The resulting files will
need to be moved to /var/lib/asterisk/keys if you want
to use them, and any private keys (.key files) will
need to be initialized at runtime either by running
Asterisk with the '-i' option, or with the 'init keys'
command once Asterisk is running.

Press ENTER to continue or ^C to cancel.
```

```
Generating SSL key 'astlabkey':
Generating RSA private key, 1024 bit long modulus
..............++++++
.....................++++++
e is 65537 (0x10001)
writing RSA key
Key creation successful.
Public key:  astlabkey.pub
Private key: astlabkey.key
```

Then, those two files (`astlabkey.pub` and `astlabkey.key`) can be copied to `/usr/lib/asterisk/keys` folder on the other SIP server. Now `/etc/asterisk/dundi.conf` file should be edited.

On `sip-server0` this content is needed:

```
;
; DUNDi configuration file
;

[general]
department=sip-server0
organization=UOC
locality=Barcelona
stateprov=BA
country=ES
email=ridelmo@uoc.com
phone=+34932532300
entityid=52:54:00:E7:EB:6C
cachetime=5
ttl=2
autokill=yes

[mappings]
registered => sipregistrations,0,IAX2,dundi:${SECRET}@sip-server0/${NUMBER
    },nopartial

[52:54:00:18:8E:30]
model = symmetric
host = 192.168.103.99
inkey = astlabkey
outkey = astlabkey
qualify = yes
include=registered
permit=registered
order=primary
dynamic=yes
```

And `sip-server1` file needs this content:

```
;
; DUNDi configuration file
;

[general]
department=sip-server1
organization=UOC
locality=Barcelona
stateprov=BA
country=ES
email=ridelmo@uoc.com
phone=+34932532300
entityid=52:54:00:18:8E:30
cachetime=5
ttl=2
autokill=yes

[mappings]
registered => sipregistrations,0,IAX2,dundi:${SECRET}@sip-server1/${NUMBER
    },nopartial

[52:54:00:E7:EB:6C]
model = symmetric
host = 192.168.103.86
inkey = astlabkey
outkey = astlabkey
qualify = yes
include=registered
permit=registered
order=primary
dynamic=yes
```

Finally, asterisk *dialplan* should be programmed, `/etc/asterisk/extensions.conf` file needs this content:

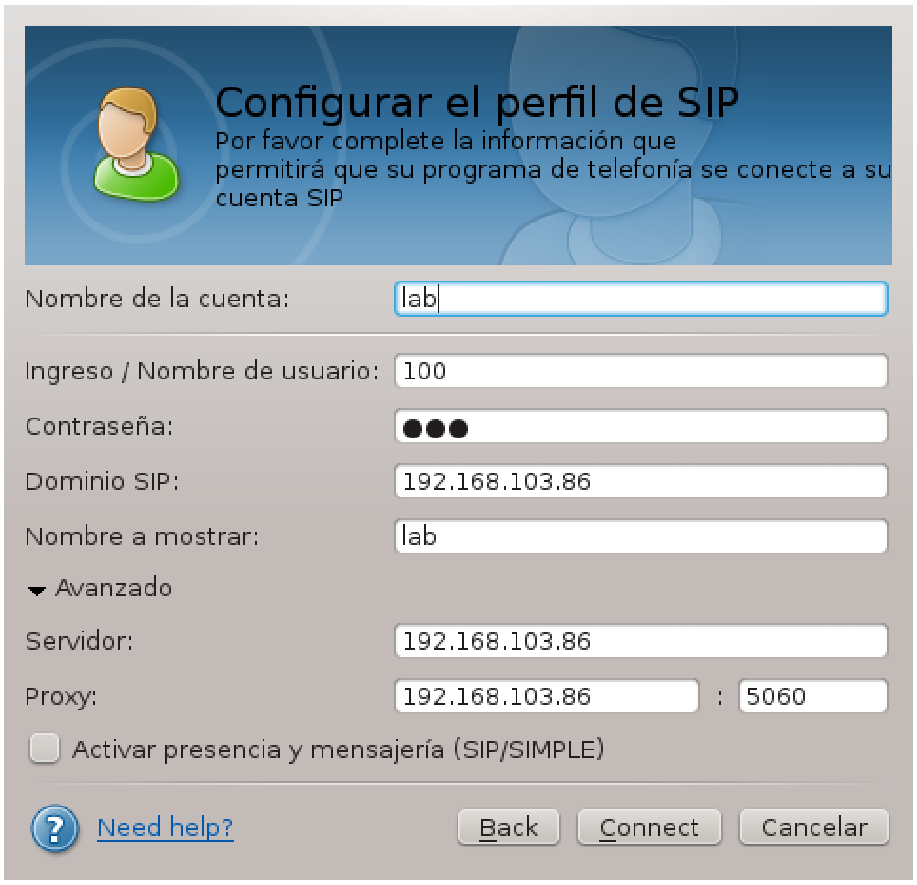
```
; extensions.conf - the Asterisk dial plan

[general]
static=yes
writeprotect=no
clearglobalvars=no

[internal]
exten => _1XX,1,ChanIsAvail(SIP/${EXTEN})
exten => _1XX,n,NoOp(${AVAILCHAN})
exten => _1XX,n,GotoIf($["${AVAILCHAN}"=""]?remote:local)
exten => _1XX,n(remote),Dial(${DUNDILOOKUP(${EXTEN},registered)})
exten => _1XX,n,Hangup
exten => _1XX,n(local),Dial(SIP/${EXTEN})
exten => _1XX,n,Hangup
```

It says that it is checked if this extension is locally registered, if that is the case it is locally dialed, if not it is searched in DUNDi peers and then, dialed.

5.5.2.3 SIP clients

As clients there are:

- a Qutecom soft phone that registers as 100 extension. At figure 5.2 it can be seen screen shot of its configuration. This soft phone doesn't support DNS SRV, so we wait until 101 extension is registered, and we register it to the other SIP server, to force both extensions are registered to different SIP registrars.



Configurar el perfil de SIP

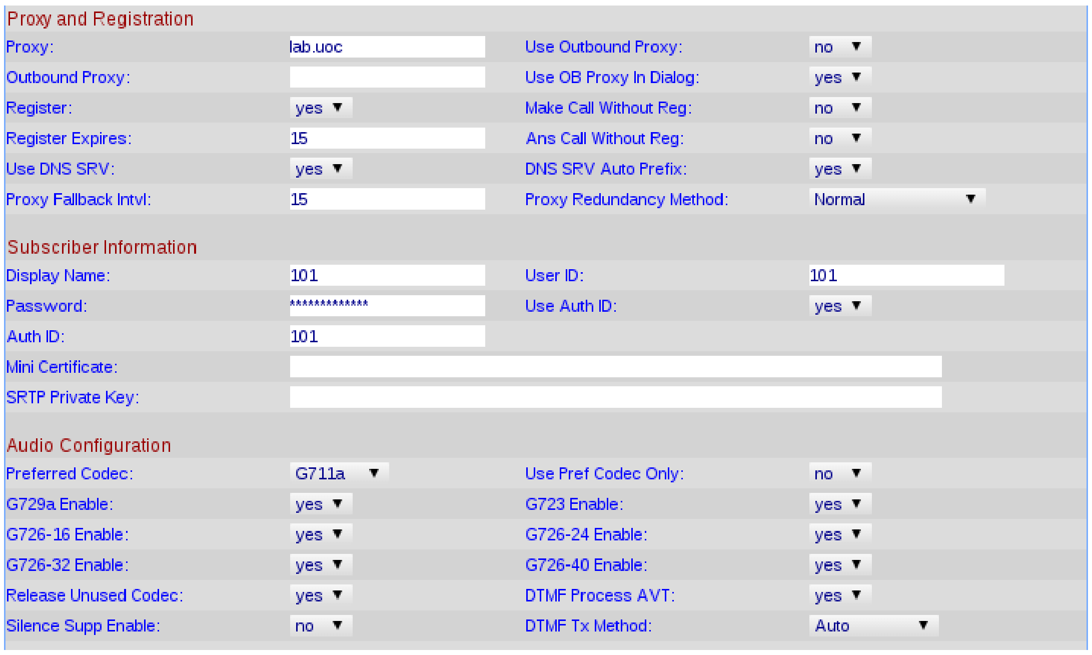
Por favor complete la información que permitirá que su programa de telefonía se conecte a su cuenta SIP

Nombre de la cuenta:	lab
Ingreso / Nombre de usuario:	100
Contraseña:	●●●
Dominio SIP:	192.168.103.86
Nombre a mostrar:	lab
Avanzado	
Servidor:	192.168.103.86
Proxy:	192.168.103.86 : 5060

☐ Activar presencia y mensajería (SIP/SIMPLE)

Figure 5.2: Qutecom configuration

- a Sipura hardware phone that registers as 101 extension. This phone supports DNS SRV an can register to any server, in case of failure, it will register to the other one. At figure 5.3 a screen shot of it's configuration can be seen.



Proxy and Registration			
Proxy:	lab.uoc	Use Outbound Proxy:	no
Outbound Proxy:		Use OB Proxy In Dialog:	yes
Register:	yes	Make Call Without Reg:	no
Register Expires:	15	Ans Call Without Reg:	no
Use DNS SRV:	yes	DNS SRV Auto Prefix:	yes
Proxy Fallback Intvl:	15	Proxy Redundancy Method:	Normal
Subscriber Information			
Display Name:	101	User ID:	101
Password:	*************	Use Auth ID:	yes
Auth ID:	101		
Mini Certificate:			
SRTP Private Key:			
Audio Configuration			
Preferred Codec:	G711a	Use Pref Codec Only:	no
G729a Enable:	yes	G723 Enable:	yes
G726-16 Enable:	yes	G726-24 Enable:	yes
G726-32 Enable:	yes	G726-40 Enable:	yes
Release Unused Codec:	yes	DTMF Process AVT:	yes
Silence Supp Enable:	no	DTMF Tx Method:	Auto

Figure 5.3: Sipura phone configuration

After this configurations are made, 100 extension is registered at `sip-server0` and 101 extension is registered at `sip-server1`.

5.5.3 Failure test

In normal operation, when a call is sent from 100 to 101 extension, it is seen this in `sip-server0` asterisk console:

```
 == Using SIP RTP CoS mark 5
   -- Executing [101@internal:1] ChanIsAvail("SIP/100-00000011", "SIP
      /101") in new stack
   -- Executing [101@internal:2] NoOp("SIP/100-00000011", "") in new stack
   -- Executing [101@internal:3] GotoIf("SIP/100-00000011", "1?remote:
      local") in new stack
   -- Goto (internal,101,4)
   -- Executing [101@internal:4] Dial("SIP/100-00000011", "IAX2/dundi:
      T8LKCzI9HzZrCAl20KWCTw==@sip-server1/101") in new stack
   -- Called IAX2/dundi:T8LKCzI9HzZrCAl20KWCTw==@sip-server1/101
   -- Call accepted by 192.168.103.99 (format alaw)
   -- Format for call is alaw
   -- IAX2/dundi-468 is ringing
   -- IAX2/dundi-468 answered SIP/100-00000011
```

```
    -- Hungup 'IAX2/dundi-468'
  == Spawn extension (internal, 101, 4) exited non-zero on 'SIP
     /100-00000011'
```

And this on `sip-server1` console:

```
    -- Accepting AUTHENTICATED call from 192.168.103.86:
       > requested format = alaw,
       > requested prefs = (alaw),
       > actual format = alaw,
       > host prefs = (alaw),
       > priority = mine
    -- Executing [101@internal:1] ChanIsAvail("IAX2/dundi-627", "SIP/101")
       in new stack
  == Using SIP RTP CoS mark 5
    -- Executing [101@internal:2] NoOp("IAX2/dundi-627", "SIP/101-0000000c
       ") in new stack
    -- Executing [101@internal:3] GotoIf("IAX2/dundi-627", "0?remote:local
       ") in new stack
    -- Goto (internal,101,6)
    -- Executing [101@internal:6] Dial("IAX2/dundi-627", "SIP/101") in new
       stack
  == Using SIP RTP CoS mark 5
    -- Called SIP/101
    -- SIP/101-0000000d is ringing
    -- SIP/101-0000000d answered IAX2/dundi-627
  == Spawn extension (internal, 101, 6) exited non-zero on 'IAX2/dundi-627'
    -- Hungup 'IAX2/dundi-627'
```

The call has followed this flow:

- it has been initiated on Qutecom soft phone by SIP protocol to `sip-server0`, where 100 extension is registered.
- then, is has been sent by IAX protocol from `sip-server0` to `sip-server1`, where 101 extension is registered.
- Then, `sip-server1` has sent the call to Sipura phone using SIP protocol, where the call has been answered and then, hung up.

Now `sip-server1` is going to be turned down, Sipura phone will register to `sip-server0` and a new call will be placed.

After `sip-server1` shuts down, it can be seen on `sip-server0` console that `sip-server1` has gone, and that 101 extension is registered to its new SIP server:

```
[Dec 23 18:29:38] NOTICE[23019]: pbx_dundi.c:2973 destroy_trans: Peer
   '52:54:00:18:8e:30' has become UNREACHABLE!
```

```
    -- Registered SIP '101' at 192.168.103.90:5060
    -- Added extension '101' priority 1 to sipregistrations
```

And now, if a new call is made, all works just with `sip-server0` up. This is shown on the console:

```
  == Using SIP RTP CoS mark 5
    -- Executing [101@internal:1] ChanIsAvail("SIP/100-00000012", "SIP
       /101") in new stack
  == Using SIP RTP CoS mark 5
    -- Executing [101@internal:2] NoOp("SIP/100-00000012", "SIP
       /101-00000013") in new stack
    -- Executing [101@internal:3] GotoIf("SIP/100-00000012", "0?remote:
       local") in new stack
    -- Goto (internal,101,6)
    -- Executing [101@internal:6] Dial("SIP/100-00000012", "SIP/101") in
       new stack
  == Using SIP RTP CoS mark 5
    -- Called SIP/101
    -- SIP/101-00000014 is ringing
    -- SIP/101-00000014 answered SIP/100-00000012
    -- Remotely bridging SIP/100-00000012 and SIP/101-00000014
  == Spawn extension (internal, 101, 6) exited non-zero on 'SIP
     /100-00000012'
```

There has been a maximum re-registration time of 15 seconds, because Sipura phone is configured to re-register every 15 seconds. So, in less than 15 second,s service is restored, but running calls will be hung up.

Chapter 6

Conclusions

There has been a look at some mechanisms to provide High Availability to our services. Normally just one technique is not enough to provide it, in order to achieve a good number of *nines*, instead more than one should be used.

Application data is anywhere. This anywhere must be highly available, so a storage solution must be used. Also those servers are connected to a network to provide a service. This connection may need to be highly available so an high available network should be used. And may be also the Operating System should be always up and running, so a IaaS service can be used. Lastly, may be needed a geographically redundant service. In this case some work on application layer may be needed.

To summarize, in real life, a combination of those methods should be used to achieve a low levels of service disruption (RTO) and data loss (RPO). There are standard ways to provide High Availability, but custom solutions at application level can be cheaper to provide a no disruption sensation.

There are in GNU/Linux, kernel modules and applications to provide all kind of those methods. It is up to us to use those open source applications to provide a fault tolerant service on top of GNU/Linux Operating System.

This paper is primary theoretical, but proof of concept complex examples have been shown, so High Availability is not theoretical in GNU/Linux, but real since a lot of time ago. As examples, they have focused only on high availability, in real production systems security must be a concern. Security at application configuration, communication between servers and clients and physical protection.

Bibliography

[1] aoetools.sourceforge.net. ATA over Ethernet Tools
http://aoetools.sourceforge.net/.

[2] apache.org. Apache CloudStack Documentation
http://cloudstack.apache.org/docs/en-US/index.html.

[3] apache.org. The Apache Cassandra project
http://cassandra.apache.org/.

[4] apache.org. The Apache Hadoop project
http://hadoop.apache.org/.

[5] Cisco. Designing High-Availability Enterprise Networks
http://www.ciscopress.com/articles/article.asp?p=375501&seqNum=2.

[6] Citrix. The three levels of high availability – Balancing priorities and cost
http://www.networkworld.com/chatterbox/citrix/dimension3/3_levels_high_availability_WP_final.pdf.

[7] clusterlabs.org. Pacemaker
http://clusterlabs.org/wiki/Pacemaker.

[8] Corosync. Corosync
http://corosync.github.io/corosync/.

[9] Davis, Thomas. Linux Ethernet Bonding Driver HOWTO
https://www.kernel.org/doc/Documentation/networking/bonding.txt.

[10] DRBD. DRBD: About
http://www.drbd.org/docs/about/.

[11] DRBD. DRBD Features: Replication Modes
http://www.drbd.org/users-guide/s-replication-protocols.html.

[12] Dworschak, Roland. LBVM . Load Balancing of virtual Machines
http://lbvm.sourceforge.net/docu/.

[13] Enhance Technology Inc. RAID Comparison Chart and Enhance Technology Storage Systems
http://www.enhance-tech.com/press/raid-comparison-and-storage-systems.html.

[14] openQRM enterprise. openQRM enterprise
http://www.openqrm-enterprise.com.

[15] eucalyptus.com. Eucalyptus Cloud Documentation
http://www.eucalyptus.com/docs.

[16] Goirand, Thomas. tgt missing init script for tgtd - Debian Bug report logs
http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=577925.

[17] ieee802.org. IEEE 802.3ad Link Aggregation (LAG) - What it is, and what it is not
http://www.ieee802.org/3/hssg/public/apr07/frazier_01_0407.pdf.

[18] ImageStream. VRRP - Virtual Router Redundancy Protocol
http://sourceforge.net/p/vrrpd/wiki/Home/.

[19] Inlab. Balance
http://www.inlab.de/balance.html.

[20] ISC. A Basic Guide to Configuring DHCP Failover
https://kb.isc.org/article/AA-00502/31.

[21] keepalived.org. Keepalived for linux
http://www.keepalived.org.

[22] kernel.org. Dracut wiki
https://dracut.wiki.kernel.org/index.php/Main_Page.

[23] linux ha.org. Heartbeat - Linux-HA
http://linux-ha.org/wiki/Heartbeat.

[24] linuxvirtualserver.org. The Linux Virtual Server Project
http://www.linuxvirtualserver.org/.

[25] Natário, Rui. Networks and Servers: High Availability - Objectives
http://networksandservers.blogspot.com.es/2011/02/high-availability-objectives.
html.

[26] Neira Ayuso, Pablo. The Conntrack tools user manual
http://conntrack-tools.netfilter.org/manual.html.

[27] NetApp. NetApp Delivers Enterprise-Class Availability
http://www.netapp.com/us/system/pdf-reader.aspx?m=ds-3100.pdf&cc=us.

[28] opennebula.org. OpenNebula - Flexible Enterprise Cloud Made Simple
http://opennebula.org/documentation:rel4.4.

[29] openstack.org. OpenStack Docs
http://docs.openstack.org.

[30] openvz.org. Quick Installation - OpenVZ Linux Containers Wiki
http://openvz.org/Quick_installation.

[31] Oracle. Determining Your High Availability Requirements
http://docs.oracle.com/cd/B28359_01/server.111/b28281/hadesign.htm.

[32] Oracle. MySQL::MySQL 5.6 Reference Manual::Replication
http://dev.mysql.com/doc/refman/5.6/en/replication.html.

[33] Oracle. Project: OCFS2
https://oss.oracle.com/projects/ocfs2/.

[34] Perforce. High availability disaster recovery solutions
http://www.perforce.com/sites/default/files/pdf/perforce-high-availability-disast
0.pdf.

[35] Preston, Tim. BIRD Route Server at LINX
http://www.uknof.org.uk/uknof15/Preston-Routeserver.pdf.

[36] quagga.net. Quagga Software Routing Suite
http://www.nongnu.org/quagga/.

[37] RedHat. Global File System 2 - Red Hat Customer Portal
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_
Linux/6/html-single/Global_File_System_2/index.html.

[38] RedHat. RedHat Cluster Suite Introduction
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_
Linux/5/html/Cluster_Suite_Overview/s1-rhcs-intro-CSO.html.

[39] stgt.sourceforge.net. Linux SCSI target framework
http://stgt.sourceforge.net/.

[40] Sun Microsystems. Enterprise Network Design Patterns
http://www.informit.com/articles/article.aspx?p=169544.

[41] Surý, Ondřej. The BIRD Internet Routing daemon
http://bird.network.cz.

[42] Wall Street Journal. Microsoft Quietly Gains Share in Virtualization
http://blogs.wsj.com/digits/2013/05/31/microsoft-quietly-gains-share-in-virtualiz

[43] wikipedia.org. Border Gateway Protocol
http://en.wikipedia.org/wiki/Border_Gateway_Protocol.

[44] wikipedia.org. Cloud Computing
http://en.wikipedia.org/wiki/Cloud_Computing.

[45] wikipedia.org. Common Address Redundancy Protocol
http://en.wikipedia.org/wiki/Common_Address_Redundancy_Protocol.

[46] wikipedia.org. Comparison of application servers
http://en.wikipedia.org/wiki/Comparison_of_application_servers.

[47] wikipedia.org. Domain Name System
http://en.wikipedia.org/wiki/Domain_Name_System.

[48] wikipedia.org. Mean time between failures
http://en.wikipedia.org/wiki/Mean_time_between_failures.

[49] wikipedia.org. Mean time to repair
`http://en.wikipedia.org/wiki/Mean_time_to_repair`.

[50] wikipedia.org. MX record
`http://en.wikipedia.org/wiki/MX_record`.

[51] wikipedia.org. Network File System
`http://en.wikipedia.org/wiki/Network_File_System`.

[52] wikipedia.org. Network Message Block
`http://en.wikipedia.org/wiki/Cifs`.

[53] wikipedia.org. Open Shortest Path Protocol
`http://en.wikipedia.org/wiki/Open_Shortest_Path_First`.

[54] wikipedia.org. Preboot Execution Environment
`http://en.wikipedia.org/wiki/Preboot_Execution_Environment`.

[55] wikipedia.org. Routing Protocol
`http://en.wikipedia.org/wiki/Routing_protocol`.

[56] wikipedia.org. Server Farm
`http://en.wikipedia.org/wiki/Server_farm`.

[57] wikipedia.org. Service Level Agreement
`http://en.wikipedia.org/wiki/Service_level_agreement`.

[58] wikipedia.org. SRV record
`http://en.wikipedia.org/wiki/SRV_record`.

[59] wikipedia.org. Virtual Router Redundancy Protocol
`http://en.wikipedia.org/wiki/Virtual_Router_Redundancy_Protocol`.

[60] wikipedia.org. Wake-On-Lan
`http://en.wikipedia.org/wiki/Wake-on-LAN`.

[61] wikipedia.org. WebDAV
`http://en.wikipedia.org/wiki/WebDAV`.